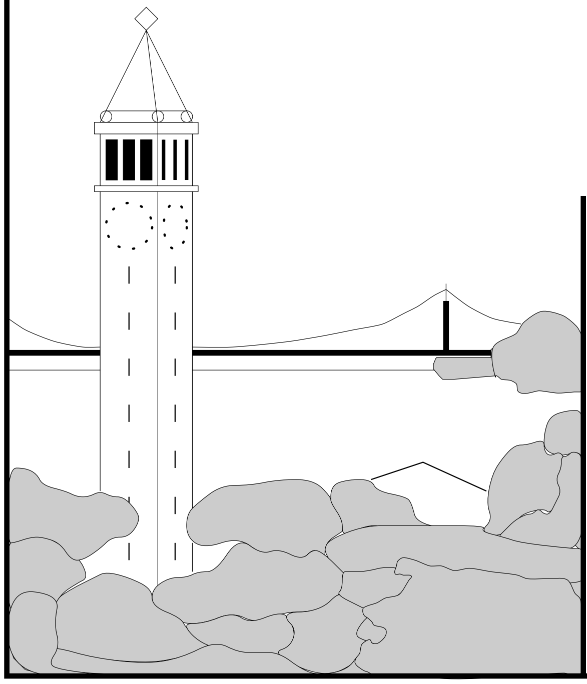
Supporting Rapid Mobility via Locality in an Overlay Network

Ben Y. Zhao, Anthony D. Joseph and John Kubiatowicz
Computer Science Division
University of California, Berkeley
{ravenben, adj, kubi}@cs.berkeley.edu



Report No. UCB/CSD-02-1216

November 2002

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

Supporting Rapid Mobility via Locality in an Overlay Network

Ben Y. Zhao, Anthony D. Joseph and John Kubiatowicz
Computer Science Division
University of California, Berkeley
{ravenben, adj, kubi}@cs.berkeley.edu

November 2002

Abstract

In this paper, we present *Mobile Tapestry*, an extension to the Tapestry overlay network protocol, that enables scalable, fault-tolerant, and timely delivery of network messages, including multimedia streams, to and from rapidly moving nodes. Mobile Tapestry efficiently supports individual mobile nodes and, by using an approach we call *hierarchical mobility*, it also supports large groups of mobile nodes simultaneously moving together. Mobile Tapestry leverages the Tapestry's locality mechanisms to reduce the latency and bandwidth of mobility update traffic, while eliminating the routing inefficiencies and availability problems of IP-based mobility protocols, such as Mobile IP. Our simulation results show that Mobile Tapestry significantly outperforms Mobile IP by providing lower latency and higher fault tolerance for message delivery.

1 Introduction

Economies of scale and technological advancements are leading to the widespread availability and use of millions of wirelessly-enabled mobile computers, Personal Digital Assistants (PDAs), IP-capable cellular phones, and other portable devices. The same economic and technological trends are also resulting in the large-scale deployment of publically acessible packet-switched wireless network access points, such as IEEE 802.11b, in both fixed (*e.g.*, hotel, airport lounge, coffee shop, neighborhood, etc.) and mobile (*e.g.*, train, airplane, ship, etc.) environments.

These wireless networks are used for traditional applications, such as web browsing and e-mail, but increasingly they are being used for latency, bandwidth, and jitter sensitive applications, including instant messaging and streamed multimedia content delivery.

We consider two different types of rapid mobility scenarios. The first type is rapid individual mobility across radio network cells (*e.g.*, a mobile user on an inter-city bus travelling on a highway with cell sizes of half a mile). At each cell crossing the mobile user's location must be updated and any in-flight messages or media content must be appropriately routed to their new location. While the total number of cell crossings per minute is relatively small, they occur frequently, yielding short cell residency times and making it difficult for mobility protocols to track and route to the mobile user.

For the second type, consider a high-speed bullet train with 1,000 mobile users. With cell sizes of half a mile, there are frequent, huge bursts of cell crossings that will overwhelm most mobility and application-level protocols. Our proposed solution, *hierarchical mobility*, yields a switching rate of only a few cell crossings per minute by placing an access point on the train and then bulk-switching a "trunk" connection between the train's access point and fixed infrastructure access points.

The challenge across both of these scenarios is *scalable* content delivery (messages and multimedia streams) to and from very large numbers of users with simultaneous *rapid* mobility of both users and network access points. With the upcoming deployment of circuit- and packet-switched 3G and all IP 4G digital cellular networks, we expect that these challenges will grow in scope and scale, as the numbers of access points, users, devices, and applications increase.

The most common related mobile routing protocol, Mobile IP [7, 9, 10], suffers from routing performance problems due to inefficient, asymmetric, triangle routing of messages when *Route Optimization* is not used (the typical case) and is also vulnerable to fault-tolerance problems due to its dependence on reachability and availability of a home agent in the user's home network.

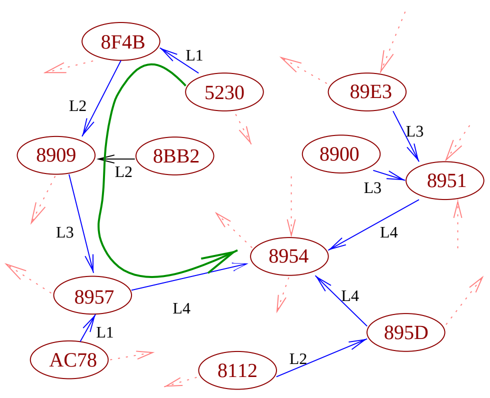


Figure 1: *Tapestry routing example.* The path taken by a message originating from node 5230 destined for node 8954 in a Tapestry mesh.

In this paper, we add mobility extensions that address the challenges of rapid mobility to Tapestry [22], an overlay network protocol. The resulting protocol, *Mobile Tapestry*, leverages and augments Tapestry's inherent scalable, fault-tolerant, locality-based routing mechanisms to provide the following contributions: scalable, fault-tolerant delivery of messages and streams to/from mobile nodes, support for rapidly moving mobile nodes using direct connections to fixed infrastructure, and support for hierarchical mobility using indirect connections through local, mobile access points with trunk connections to fixed infrastructures.

The rest of the paper is organized as follows: we first present background information about the Tapestry overlay network mechanisms in Section 2. In Section 3, we present the the algorithms for basic mobility support in Mobile Tapestry, followed by a discussion in Section 4 of how mobile nodes publish and update their location bindings efficiently. We then present results from simulations, in Section 5. Finally, we present related work in Section 6 and conclude in Section 7.

2 Tapestry Routing and Location

In this section, we review Tapestry's operation and benefits. Tapestry is a fault-tolerant, overlay/data location and routing layer [22] that extends a similar hashed-suffix mesh mechanism introduced by Plaxton, Rajaraman and Richa (PRR) in [11], by supporting dynamic construction of the network using only decentralized knowledge. Tapestry is novel in allowing 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. Tapestry provides delivery time within a small factor of optimal, between any two points in the network.

Each Tapestry node acts as one or a combination of a *server* (where objects are stored), *router* (which forward messages), or *client* (origins of requests). Objects and nodes are named in a both location- and semantic property-independent manner using random fixed-length bit-sequences with a common radix – for example, 40 hexadecimal digits representing *Globally Unique IDentifiers* (GUIDs) of 160 bits. We will use a uniform notation to say that node N has GUID $\texttt{N}_G$ and object O has GUID $\texttt{O}_G$. The system assumes entries are roughly evenly distributed in both node and object namespaces, which can be achieved by using the output of a secure hashing algorithm like SHA-1 [16].

2.1 Routing Layer

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 used by CIDR IP address allocation [13]. 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, and contains a number of entries equal to the base of the ID. The i^{th} entry in the j^{th} level is the ID and location of the closest node that begins with prefix($N, j-1$)+"i" (*e.g.*, the 9^{th} entry of the 4^{th} level for node 325AE is the node closest to $\underline{\texttt{325}}$AE

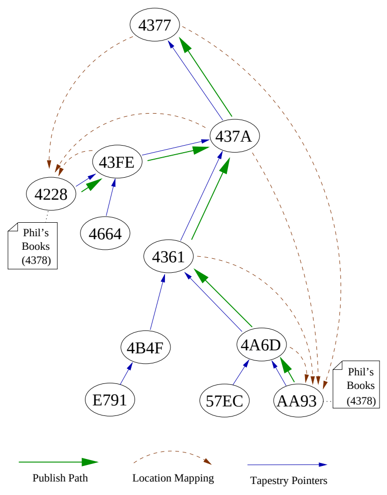


Figure 2: *Tapestry object publish example.* Two copies of an object (`4378`) being published towards the object's Tapestry location root at `4377`.

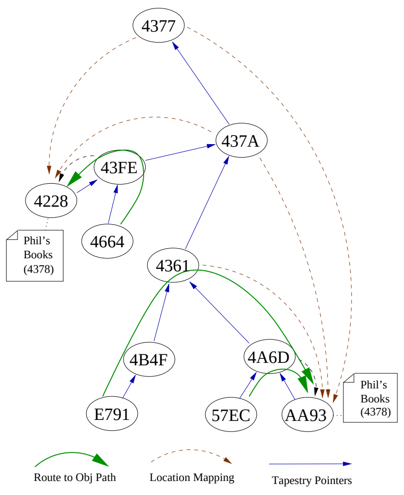


Figure 3: *Tapestry route to object example.* Several nodes send messages to object `4378` from different points in the network. The messages route towards the object's Tapestry location root, and when they intersect the publication path, they follow location mappings to the nearest copy of the object.

in network distance that begins with `3259`[1].

When routing, the n^{th} hop shares a prefix of at least length n with the destination ID. To find the next router, Tapestry looks in its $(n+1)^{\text{th}}$ level map for the entry matching the next digit in the destination ID. This routing method guarantees that any existing unique node in the system will be reached in at most $Log_b N$ logical hops, in a system with an `N` size namespace using IDs of base b, assuming consistent neighbor maps.

For redundancy purposes and to enable route selection based upon latency, each routing entry in the neighbor map contains c forwarding pointers to different neighbor nodes for the same digit (currently $c = 3$). Since the preceding digits all match the current node's prefix, the map only keeps a small constant size, $c \cdot b$, set of neighbor pointers at each route level for a total fixed constant size of $c \cdot b \cdot Log_b N$.

One 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 while en route to the root node. In short, the hashed-prefix mesh of neighbor maps is a large set of embedded trees in the network, one rooted at every node. Figure 1 shows an example using hexadecimal digits of length 4 (65,536 nodes in the namespace).

2.2 Data Location

Tapestry leverages this routing infrastructure in a straightforward way to perform data location. Each object is associated with one or more *Tapestry location roots* through a distributed deterministic mapping function, that when applied from any node in the network, maps a given object ID to the same resulting root node [22]. In the following we will assume that object `O` has a single root $\texttt{O}_R$; we will amend this later in Section 3.4. A server `S` that stores a copy of object `O` advertises or *publishes* this object by sending a publish message toward the Tapestry location root for that object (see Figure 2). At each node along the path to the root, the publish message stores location information in the form of a pointer mapping: $<\texttt{O}_G$, $\texttt{S}_G>$. That is only a pointer, not a copy of the object itself. When there are multiple copies of an object, each server maintaining a replica publishes its copy. Each node `N` stores location mappings for multiple replicas in sorted order of distance from `N`.

To locate an object `O`, a client sends a message to `O`'s Tapestry location root (see Figure 3). Tapestry routes the message hop-by-hop and checks at each hop if the node contains a location mapping for `O`. If a mapping exists,

[1]Earlier descriptions of Tapestry used suffix-based (*i.e.*, right to left) routing. In reality, prefix- and suffix-based routing are isomorphic; prefix routing seems easier to explain, however.

Tapestry redirects the message to the object's server. Otherwise, Tapestry forwards the message one hop closer to the root until the message reaches `O`'s root, where it is guaranteed to find a location mapping for `O`.

The hierarchical nature of Tapestry routing means that at each hop towards the root, the number of nodes satisfying the next hop constraint decreases by a factor equal to the identifier base (*e.g.*, 3 or 4 bits). Intuitively, Tapestry's locality property works as follows, by sorting the distance to multiple replicas at intermediate hops, clients searching for nearby objects are likely to quickly intersect the path taken by publish messages, resulting in their finding the *nearest* replica of the desired object.

2.3 Tapestry Benefits

Without going into the details and analysis of the algorithms [6], we summarize the key benefits of the Tapestry location and routing infrastructure:

- *Responsive Fault Handling*: Nodes use soft state beacons to quickly detect faults. Since Tapestry provides multiple paths to every destination, messages route around faults by sending duplicate packets down redundant paths, where appropriate or dictated by application-specific policies.

- *Wide-area Scalability*: Tapestry routing is inherently decentralized, using information from a number of nodes logarithmically proportional to the size of the network. Routing tables also have size logarithmically proportionally to the network size, guaranteeing scalability as the network scales.

- *Locality-biased Routing*: Tapestry routing tables contain multiple routes to neighbors sorted by physical latency. By taking latency into account, overlay routing exploits locality to prevent local communication from "escaping" into the wide-area. In a similar system [11], the network distance traveled by a message was shown to be linearly proportional to the ideal underlying network distance. In fact, previous Tapestry experiments show that proportionality is maintained within a small constant in real networks [22]. In contrast, systems such as [12, 19] are subject to undesirable scenarios where local communication traverses the globe.

3 Algorithms and Architecture

In this section, we present a detailed description of how Tapestry algorithms and mechanisms can be used to support mobile nodes and objects. We outline several mobility scenarios and describe how Mobile IP handles them. Then, we present modifications to the basic Tapestry infrastructure for mobility support. This is followed by detailed discussion of the issues of fault-tolerance and security. We will save discussions of proxy handoff and rapid mobility for Section 4.

3.1 Mobility Scenarios

In an ideal mobile system, performance should not be a function of the distance between a roaming *mobile node* (MN) and its home network. Instead, latency should be a factor of the distance between the mobile node and the node with which it is corresponding, the *correspondent host* (CH). Furthermore, the network should be aware of the mobile node's movements at all time, delivering messages to it in a timely fashion.

The performance of current mobile systems, however, is far from ideal. A mobile node's distance from the home network is the primary factor impacting performance. Furthermore, existing systems are not structured to handle rapid mobility, making persistent or interactive connections impossible to maintain.

In this paper, we consider four key mobility scenarios: (1) when the MN is close to its home network, (2) when the MN is far away from the home network but close to the CH, (3) when the MN is far away from its home network and far away from the CH, and (4) when the mobile node is moving rapidly highlighting the need for updates to propagate location information through the network in a timely manner.

3.2 Mobile Scenarios with Mobile IP

The *de facto* standard for supporting mobile hosts is Mobile IP [7, 9, 10], which uses a *Home Agent* (HA) and *Foreign Agent* (FA) to tunnel packets from a *Correspondent Host* (CH) thru the *Mobile Node's* (MN) home network (using

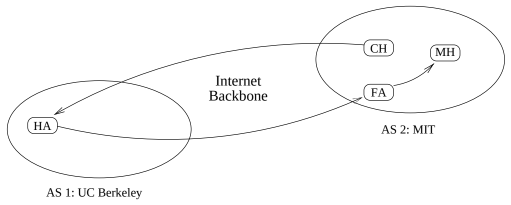


Figure 4: *A Mobile IP scenario.* A Professor at MIT sends an instant message to his visiting colleague from Berkeley. The *correspondent host* (CH) sends a message to the *mobile host* (MH) by first routing through MH's *home agent* (HA) and *foreign agent* (FA).

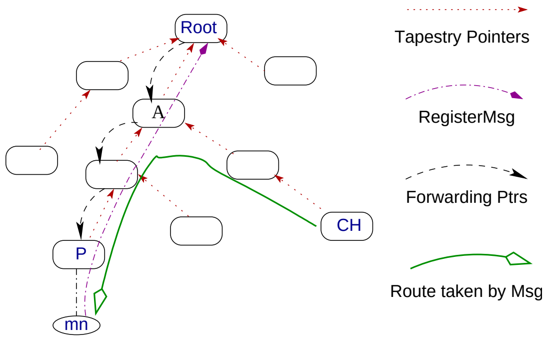


Figure 5: *Communicating with a mobile host.* Mobile node mn registers its location with a RegisterMsg, which constructs a path of backpointers from the root node of mn to proxy P. Correspondent host CH sends a message to mobile node mn, which routes towards the root until it finds the path of backpointers at node A, and follows them to P and mn.

the HA) to the FA at the MN's current location[2]. A extension mechanism called *Route Optimization* [9] enables nodes corresponding with mobile nodes to cache the MN's current location information, bypassing the home agent. Note that support for mobility in IPv6 [8] incorporates Mobile IP design and algorithms. In this paper, we will treat Mobile IP and Mobile IPv6 as the same system for comparative purposes.

There are several limitations to Mobile IP. First, standard Mobile IP (without Route Optimization) requires that messages be routed to the mobile node's home agent before being tunnelled to the foreign agent or mobile node. This asymmetric *triangle routing* is clearly inefficient, and can significantly increase routing distance and latency. The example scenario in Figure 4 shows the cost of triangle routing. Second, update information needs to be propagated to several nodes, including the (potentially distant) home agent, before the location update takes effect. These two factors combine to limit the feasibility of Mobile IP support for fast moving nodes. Third, the dependence on one or more home agents limits not only scalability for networks with a large roaming population, but also provide tempting targets for denial of service attacks. Finally, the dependence on the availability of both the home agent and foreign agent means that a single server or routing failure can result in a mobile node being inaccessible from correspondent hosts.

Route optimization, a technique for removing triangle routing, causes other problems. It requires modifications to correspondent hosts, introduces non-transparency, and requires route cache coherence protocols at correspondent hosts. Non-transparency means that correspondent hosts must be aware of whether the destination of a message is a mobile node. Cache coherence protocols are a requirement, since out of date location bindings for mobile nodes can more than double the overhead of triangle routing. For these three reasons, route optimization is not normally used.

There are several fundamental design differences between mobile support in Tapestry and Mobile IP. First, *Mobile Tapestry* utilizes Tapestry location mechanisms to efficiently distribute location information for a mobile node, preserving locality such that correspondent hosts and mobile nodes can reach nearby mobile nodes quickly. Location update messages also preserve locality, such that the distance travelled is proportional to distance travelled by mobile nodes between updates. Finally, we have designed Mobile Tapestry with the belief that wide-area storage and bandwidth are plentiful, and moderate amounts of redundancy can be used to improve message reliability and performance.

[2]Note that some Mobile IP implementations support a *pop-up mode* where a MN can act as an FA.

These design choices greatly impact basic Mobile Tapestry behaviour. First, Mobile Tapestry nodes require no notion of a *home network*, and routing is equally efficient independent of location, while routing latency is roughly proportional to the network distance between the endpoints. Second, communicating with mobile nodes is completely transparent to Tapestry nodes, provided that mobile nodes set the “mobile” flag in their GUIDs[3] at the Tapestry layer. Finally, redundancy is used in location updates and message routing, where appropriate, to reduce and hide node- and network-level failures from the application layer.

3.3 Basic Mobility Support

When a mobile node moves to a location outside of its home network, messages destined for the mobile node must be forwarded to its new location. In both Mobile IP and Mobile Tapestry, mobile nodes connect to local *proxy nodes*, which act as their temporary care-of-addresses in the new network. For basic mobility support, we require not only the ability to send and receive messages at the mobile node without interruption, but location services support for mobile objects on the mobile node. Here, we present algorithms providing this functionality for mobile nodes away from their home networks. Algorithms dealing with movement, updates and rapid mobility are discussed in Section 4.

Mobile nodes in the Tapestry network are client-only Tapestry nodes, which use Tapestry to communicate with other nodes and provide location service for its objects, without routing messages or storing object pointers on behalf of other nodes. This mode of operation is reasonable, since the mobile nature of these nodes means that their active participation in Tapestry routing and integration algorithms might adversely impact overall Tapestry performance. We assume that mobile nodes can detect and communicate with nearby proxy nodes in their path. For example, they could use Dynamic Host Configuration Protocol (DHCP) discovery services [5]. Furthermore, we assume that from the perspective of mobile nodes, the Tapestry network is largely static, with infrequent addition and removal of nodes.

In this section, we present two components of basic mobility support in Tapestry, *Node Registration* and *Location Services for Mobile Objects*.

3.3.1 Node Registration

In order to establish itself as a communication endpoint, a mobile node MN with a Globally Unique IDentifier (GUID) $\mathtt{MN}_G$ initially registers itself with a nearby proxy P with a registration message signed by its private key, K_{MN}: $<\mathtt{MN}_G, \mathtt{P}_G, \text{nonce}>_{K_{MN}}$. Note that as with all Tapestry nodes, GUIDs are created by applying a secure hashing function to a public key. An additional security mechanism would be to require a signed certificate showing a trusted chain between the incoming GUID and a trusted certificate server. By using the node’s public key to create its GUID, Tapestry prevents spoofing of a node GUID. This and other security techniques are discussed in Section 3.4. Mobile node GUIDs have a special tag that marks them as mobile nodes, so that they are treated differently from fixed nodes by Tapestry routers.

A proxy node treats the RegisterMessage as a special type of object publish message, and forwards it to one or more dynamically but deterministically mapped root nodes of MN. At each node along the publication path from the proxy to the root node, the RegisterMessage is used to store a backpointer to the previous hop node on the publication path. Note that this is in contrast to normal Tapestry location, which stores a location mapping from the object ID to the ID of its location. The result is a *forwarding path* of single hop backpointers from the root node to the current proxy node. When another node, CH, sends a message to MN, Tapestry routes the message towards the root node of MN. If the message intersects the forwarding path, it follows the more direct path of backpointers to the proxy, and then to the mobile node. Otherwise, the message reaches the root node and is then routed along the path of backpointers to the proxy. Note that communication from a mobile node is identical to that from a static Tapestry node once it reaches the mobile proxy.

Figure 5 shows the process of initial registration, an initial RegisterMsg setting up forwarding routes up to the root node, and a message being routed from a correspondent host, CH, to the mobile node.

Mobile Tapestry handles agent/proxy discovery by relying on similar mechanisms to those used by Mobile IP, where mobile nodes listen for periodic broadcast announcements from nearby proxy nodes. Fast-moving nodes can proactively solicit proxy nodes via expanding ring search multicast/broadcast-based messages [2] to reduce discovery latency.

[3]Tags are an optional 4-bit sequence, following the 160-bit GUID, that is used for application-specific information.

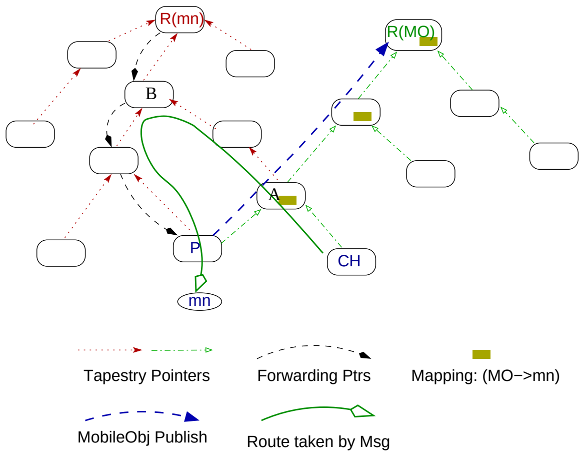


Figure 6: *Access to objects on a mobile node.* Correspondent host, `CH`, sends a message to object `MO` residing on mobile node `mn` with proxy `P`. The message is first routed to root node of `MO`. When the message locates a mapping of `MO` to `mn` at `A`, it then routes to the root node of `mn`, finding the path of backpointers at `B`, and follows them to locate `P` and `mn`.

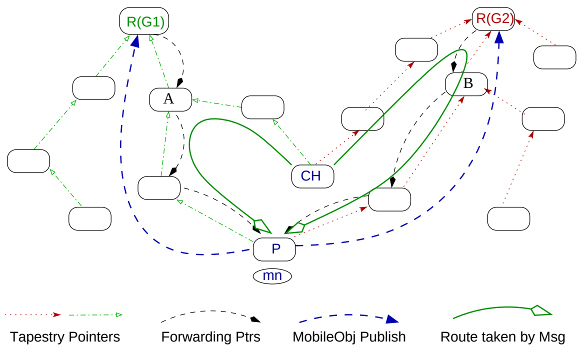


Figure 7: *GUID aliasing with 2 GUIDs.* This shows a correspondent host (`CH`) communicating to a mobile host (`MH`) using GUID aliasing. The mobile host hashes its GUID with 2 natural numbers, and registers with the resulting GUIDs $\texttt{mn}_{G1}$ and $\texttt{mn}_{G2}$. `CH` first sends a message in parallel to `MH` using both GUIDs, and caches the closer route using $\texttt{mn}_{G1}$ for future communication.

3.3.2 Location Services for Mobile Objects

In addition to supporting direct point to point communication between mobile nodes, correspondent hosts, and other mobile nodes, Mobile Tapestry also supports mobile nodes that act as servers making objects, `MO`, on mobile nodes available as if they were on fixed Tapestry nodes. We enable location services for mobile objects by introducing an additional level of indirection in the address of the published object `MO`. Mobile Tapestry publishes the address of `MO` as a mobile address. A message first routes towards the root node of `MO`, $\texttt{MO}_R$. When Mobile Tapestry identifies `MO`'s address as being encoded as a mobile GUID (using the mobile tag), it routes the message towards the root node of the mobile GUID, and intersects the publication path at or before the root node. Finally, the message traverses the publication backpointers to reach the proxy for `mn`.

Figure 6 shows the path taken by a message being routed from a correspondent host, `CH`, to a mobile object, `MO`, residing on a mobile node `mn`. The message first routes to the root node of `MO`. En route it will find the location mapping of the object, which is the Mobile Tapestry GUID for `mn`. Tapestry then routes this message to the root node of `mn`'s GUID, finding and following backpointers along the way and reaching `mn` where the object resides.

3.4 Fault-Tolerance and Security

In this section, we include brief summaries of our approaches towards fault-tolerance and security; the details are omitted for length considerations. To make Mobile Tapestry more resilient to link and node failures, we take several approaches to add redundancy to the basic design. First, we leverage existing fault-tolerant mechanisms in Tapestry, including redundancy in the routing mesh and link and router failure detection, discussed in detail in [22]. Next, when mobile nodes are within range of multiple proxy nodes, they can register with multiple proxies, and receive messages via the shortest path or via multiple paths for additional redundancy. Finally, forwarding paths for mobile nodes expire periodically. Mobile nodes periodically re-register, refreshing the forward path and helping it to dynamically adapt to failed links and nodes.

Mobile Tapestry's security mechanisms focus on preventing the spoofing of mobile GUIDs, and Denial of Service (DoS) attacks based on flooding the system with mobile registrations of forged identities. The security mechanisms rely on underlying individual Tapestry links authenticated via MACs. Spoofing to hijack connections and forging control messages are prevented by using a timestamped challenge response system, utilizing a public/private key system. We also make multiple identity attacks [4] more difficult by using a combination of signed certificates for authorization and requiring the solving of brute force problems per registration.

3.5 GUID Aliasing

An interesting mechanism for fault-tolerance and improved performance is to have the mobile node `mn` advertise its presence via multiple roots (by registering multiple GUIDs). We call this mechanism *GUID aliasing*, a variant of which provides multiple roots per object in normal Tapestry location. In GUID aliasing, the mobile node hashes its original GUID along with sequential natural numbers to generate a small set of randomized GUIDs, and registers each as its own. These GUIDs each create a forwarding path to the mobile proxy via a distinct and independent root node.

When establishing a connection, a correspondent host (`CH`) generates these GUIDs independently, and sends messages in parallel on all forwarding paths. With feedback from the mobile node, `CH` can choose the GUID that incurs the least latency for their connection. By choosing to send packets on the forwarding path with smallest latency, `CH` effectively reduces message delivery time to that of the shortest forwarding path between it and the mobile node. Figure 7 shows how a correspondent host starts communication with mobile host with GUID aliasing factor of two.

Alternatively, the CH can choose to continue to send duplicate messages out to several forwarding paths for additional fault-tolerance. This tradeoff of bandwidth resources for fault-tolerance, performance, and performance stability is explored further in Section 5. We show in Section 5.3 that two GUIDs are enough to gain significant improvement in performance and performance stability. As a less bandwidth-intensive alternative, a correspondent host can choose to send messages using a single salted GUID (or a subset of the full set), and if a link or node failure is detected, the node can then replicate and multiplex the message to additional other salted GUIDs en route.

4 Supporting Rapid Mobility

In this section we discuss the features of Mobile Tapestry which deal with movement, updates, and rapid mobility. First, we present the basic proxy handover algorithm for propagating location binding updates. The we give a short intuition of why these location updates are scalable, and how that scalability facilitates support for rapid mobility. We then present our definition of *hierarchical mobility*, and show how levels of indirection in Mobile Tapestry can be leveraged to support this type of mobile operation. This is followed by an overview of indirection in Mobile Tapestry, followed by some extensions to fault-tolerance and security.

4.1 Proxy Handover

As a mobile node moves across the coverage areas of different proxy nodes, it informs the network of its current location by sending a ProxyHandoverMsg, $<\texttt{MN}_G, \texttt{oldP}_G, \texttt{newP}_G, \text{nonce}>_{K_{MN}}$, where $\texttt{oldP}_G$ and $\texttt{newP}_G$ are the GUIDs of the old and new proxies, respectively. In areas of overlapping coverage, `mn` performs a handover from `oldP` to `newP` by sending a ProxyHandoverMsg to `newP`. `newP` sets up a forwarding route to `mn`, and requests that `oldP` set up a forwarding pointer to `newP`. `newP` then forwards the ProxyHandoverMsg on the route to the root node, $\texttt{MN}_R$,

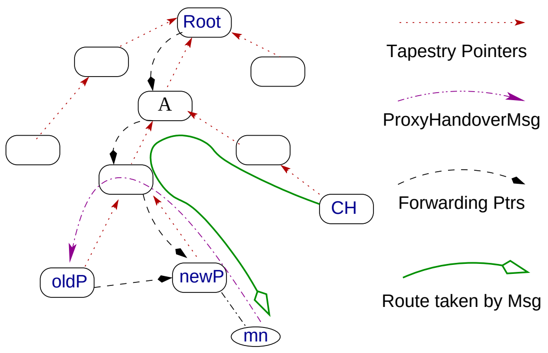


Figure 8: *Updating a location binding via ProxyHandoverMsg.* Correspondent host `CH` sends a message to mobile node `mn` after `mn` updates the Mobile Tapestry on its movement from proxy `oldP` to proxy `newP`. The ProxyHandoverMsg routes towards root of `mn` until it intersects the old path at node `A`, and backtracks to `oldP`, removing old forwarding pointers along the way.

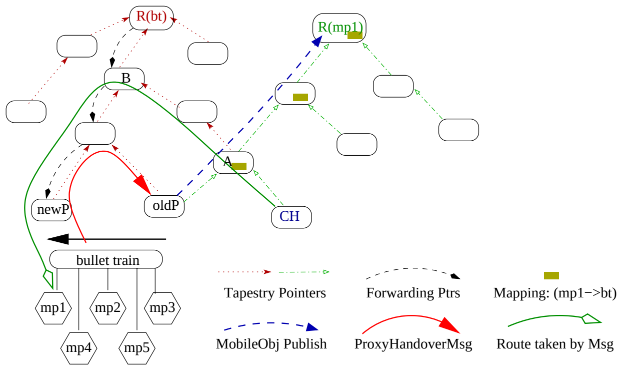


Figure 9: *An example of hierarchical mobility*. Five mobile passengers (`mp1..5`) are connected wirelessly to the network server of the bullet train (`bt`) they ride. As the bullet train moves and updates its location binding from `oldP` proxy to `newP` proxy, another node `CH` sends a message to mobile passenger 1 (`mp1`). The message routes to root of `mp1`, and finds a mapping at `A`. It starts routing towards root of `bt`, and finds the path of backpointers at `B`.

and backpointers are stored at each node along the path to the root. The message is forwarded until the new path either intersects the previously published *forwarding path* or reaches $\mathtt{MN}_R$.

When the message reaches a node `A` that is either on the previous forwarding path or the $\mathtt{MN}_R$, the node redirects its forwarding pointers to point to the new path. The node `A` forwards the message downwards to `oldP`. Each node along the downward path schedules its forwarding pointer for deletion and forwards the message towards `oldP`[4]. When the ProxyHandoverMsg reaches at `oldP`, `oldP` schedules the forwarding pointer to `newP` for deletion. Once all old forwarding pointers are deleted, handover is complete. Figure 8 shows an example of this algorithm.

If `oldP` and `newP` do not overlap in their coverage area, then `mn` will have a window of time after it leaves coverage of `oldP` and before it completes handover to `newP`. In this scenario, `oldP` performs a limited amount of buffering for `mn`, and then forwards the buffer to `newP` when a forwarding pointer is established [1].

To help ensure that, in both handover scenarios, no data is lost in transit, incoming messages are either forwarded along a preset route to the new proxy for `mn`, or are otherwise temporarily queued until a forwarding message is received at `oldP`.

[4]A delay in deleting forwarding pointers is required to handle potential reorderings of messages between nodes by the underlying transport layer.

4.2 Impact of Locality

To help the reader understand the main benefit of Mobile Tapestry, we give here some intuition on why location updates and mobile communication are efficient in terms of performance, bandwidth and load in Mobile Tapestry.

Recall that the key to Mobile Tapestry is that mobile nodes publish their location information via the Tapestry location layer (Section 2.2), which is known to exhibit locality properties. By locality we mean the ability to minimize message traversal across the wide-area whenever possible. Messages to objects find the closest replica to them while minimizing the path taken. While no rigorous proof exists for Tapestry, experiments show that the path taken is within a low constant factor of the ideal distance to the object [22], and a similar proof for the PRR system exists [11].

Consequently, we expect that routing to mobile nodes will exploit locality, and that the latency taken for a message to route from a correspondent host to a mobile node will be within a small factor of the shortest path latency between them. Note that location updates from a new proxy travel the same path to the old proxy as a correspondent host's message to the mobile node's previous location. Therefore, we expect location updates to also perform within a small factor of the ideal path latency between the old proxy and the new proxy.

The impact of this type of locality is that messages and location updates scale roughly proportionally with the shortest path distance. This is highly desirable. For instance, a visitor from Berkeley conversing via instant messaging with a local professor at MIT sees performance that scales with the distance between the two laptops, and not as a function of the distance he has traveled to get there. In Section 5, we will confirm these intuitions with experimental results, and show how they allow Mobile Tapestry to support rapid mobility.

4.3 Hierarchical Mobility

In addition to supporting the traditional mobility model of an independent mobile host, Mobile Tapestry also supports an additional level of indirection that we call *hierarchical mobility*. Hierarchical mobility applies to environments where a group of secondary mobile hosts are physically collocated with a moving primary mobile host, such as a large number of mobile passengers with wireless laptops and PDAs riding a bullet train, whose on-board network server is relaying wireless network access to all of its passengers. The train itself is moving, and its network server is a mobile node from the perspective of the land-based network. In this scenario, we refer to the train server as the *mobile parent* of the mobile nodes on-board, and to the mobile nodes as the *mobile children* of the train server.

In a hierarchical mobility scenario, naive mobility support would call for each mobile child to update location bindings as its travels. This causes scalability problems between the mobile children and nearby proxies, however, as synchronized updates flood each new proxy when it comes in range of the mobile parent.

Our solution is to impose another level of indirection between the mobile parent and its mobile children. The mobile parent, in this case the network server on a bullet train, advertises the IDs of its mobile children nodes as objects it stores. It publishes the GUIDs of the mobile children using Tapestry location. When a client routes a message to a mobile child (`mp1`), it searches for the GUID using Tapestry location. Instead of finding a list of backpointers to the mobile node, the search returns a mapping to another mobile GUID, that of the train server. When the message routes towards this GUID, it eventually finds the trail of backpointers, and routes to the train server, which forwards it to `mp1`. The benefit to such a design is that while mobile nodes maintain their own identity on the train, only the train server needs to inform local proxies of its movement. Therefore, hundreds or thousands of passengers on a wirelessly connected moving vehicle need only send out a single location binding update per new proxy.

It is easy to understand the scalability of this mechanism. Consider again our scenario of a high-speed bullet train carrying 1000 networked passengers. Where the storm of synchronized location updates would have disabled any foreign agent in a Mobile IP system, the same train under Mobile Tapestry would only require a single location update per proxy handover, on behalf of the train's server.

4.4 Levels of Indirection

We note that the mechanism used here is very similar to the level of indirection used to support basic mobile nodes in Tapestry. In essence, these levels of indirection allow us to support a hierarchy of types, as shown in Figure 10. A type on the n^{th} level relates to a type on the $n-1^{th}$ level as its mobile child node or an object residing on it.

Throughout this paper, we have shown how Mobile Tapestry exploits the ability to tunnel routing messages through multiple levels of indirection. The basic mobile node makes itself available to correspondent hosts by using indirection as if it were an object stored on its proxy node. Messages routing to objects on mobile nodes go through indirection

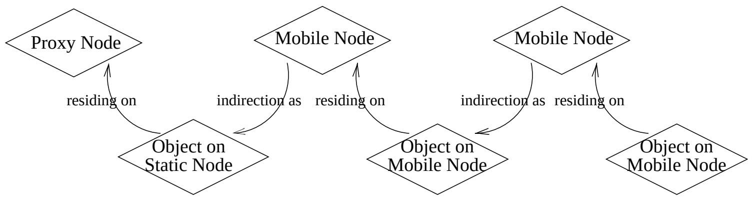


Figure 10: A figure summarizing the *levels of type indirection* used in mobile Tapestry. The arrows point out relative relationships between types.

twice: once to locate the mobile node's GUID and once to locate the mobile node's proxy. Finally, mobile nodes may make themselves available to CHs by appearing as if they were objects stored on other mobile nodes.

This indirection mechanism can be used repeatedly to produce a chain of redirections, each of which relates a node to its next level counterpart through a client - proxy relationship. For instance, this model can account for wearable computing components on the body of a passenger, exchanging information with the world through a wearable network hub on the passenger; the passenger's hub relays data to the train server, which relays to a Mobile Tapestry proxy, *etc.*

4.5 Redudant Backpointers

After introducing Mobile Tapestry's mechanisms for rapid mobility, we note an extension to our previous discussion on fault-tolerance. Backpointers in the *forwarding path* represent single points of failure, since a single link or node failure will render the mobile host unreachable. One solution is the following: as the RegisterMsg or ProxyHandoverMsg updates pointers on the forwarding path, each node stores backpointers to the next node two hops away between it and the proxy, in addition to the standard one-hop backpointer. This solution provides protection against multiple failures, except those where two or more consecutive hops on the same Tapestry path fail simultaneously. The space overhead is a factor of two over single hop pointers at internal Tapestry nodes.

5 Measurements and Evaluation

In this section, we present experimental results that evaluate the Mobile Tapestry design and architecture in terms of performance, scalability and reliability. There are several key properties of interest. First, we begin by describing our simulation framework. Then we compare the routing performance of Mobile Tapestry against that of Mobile IP. Next, we examine the impact of routing using multiple roots on performance and variability. We then explore the limits of rapid mobility for each system by simulating the message bandwidth and processing throughput requirements imposed by each algorithm on network routers. Finally, we inject artificial failures into a simulated network in order to examine performance under unreliable conditions.

5.1 Simulation Framework

Our key performance metric is *Relative Delay Penalty* (RDP) [3] applied to routing latency, between mobile nodes and randomly placed correspondent hosts. Latency RDP is the ratio of message delivery time across an overlay or redirection layer, to the delivery latency of the underlying network layer. For our RDP measurements, we used the smallest latency possible between the MN and CH as the network layer latency. In other words, we compare against the optimal (*i.e.*, shortest path) latency, not against IP latency (*i.e.*, from a path chosen by the BGP routing protocol), which is often suboptimal. Previous results show that under certain conditions, Mobile Tapestry performs as well as direct IP routing [22].

Note that the experiments do not include computational overhead for nodes (home agents or Tapestry nodes). We believe that as processing power grows with Moore's Law, overall performance in mobility systems will continue to be dominated by network latencies.

Our measurements are taken from a proprietary packet-level simulation environment, running on transit stub topologies [20] of 5,000 nodes. Each topology has 6 transit domains of 10 nodes each, and each transit node has 7 stub domains with an average of 12 nodes each. Our simulator tracks Tapestry messages across the network in logical time,

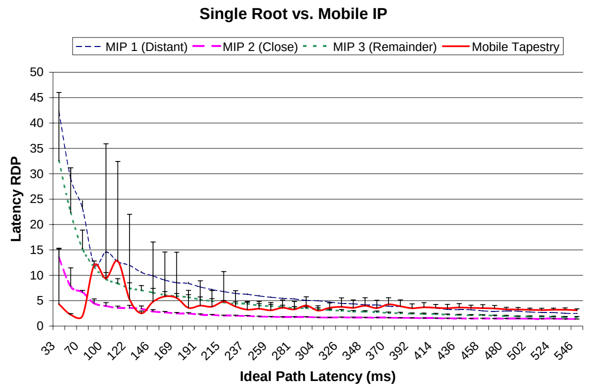


Figure 11: *Comparing routing latency overheads* of Mobile Tapestry to Mobile IP in three scenarios: (1) the MN is far from its home network and HA, (2) the MN node is near its HA, and (3) the MN is in between (1) and (2). The error bars are $\frac{1}{2}$ of the standard deviation.

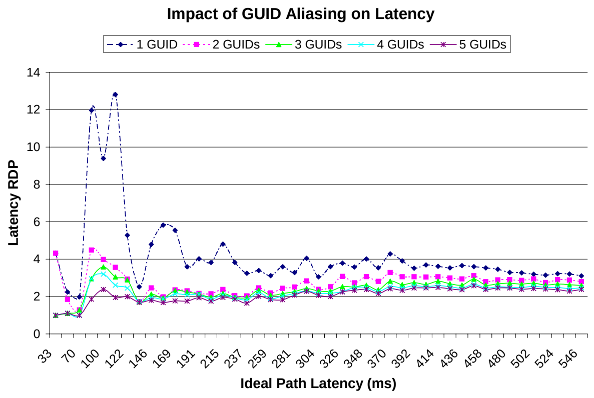


Figure 12: *Measuring the impact of additional routing roots* in Mobile Tapestry on routing latency overhead. We compare a single root to 2, 3, 4, and 5 roots.

but does not simulate network effects, such as congestion control, BGP routing policies, or packet retransmission at the transport or link layers.

Our data analysis shows that Tapestry performance varies across multiple topologies with similar transit stub configurations, and also across multiple overlay node placement configurations. This result confirms previous work that found similar variability in overlay networks [15]. To reduce such variability, we performed our measurements on 9 different 5,000 node transit stub topologies, each with 3 random overlay assignments, and collated the results.

5.2 Routing Efficiency

We studied the relative routing performance of Mobile Tapestry and Mobile IP under different roaming scenarios. In each scenario, a CH routes messages via the redirection layer to a mobile node. Mobile IP performance is a function of both the distance between CH and MN, and the distance between the MN and its HA. Mobile Tapestry, in contrast, has no notion of a home network, and routing latency is dependent solely on the distance between the CH and MN. Thus, for a more appropriate comparison, we use three Mobile IP scenarios: (1) the MN is near the HA, (2) the MN is more than two-thirds the diameter of the network away from the HA, and (3) the MN is in between (1) and (2).

In Figure 11, we plot the RDP of the three Mobile IP scenarios and basic Mobile Tapestry. The results show that for low ideal path latencies (*i.e.*, MN is near CH), Mobile IP generally performs quite poorly under scenarios 1 and 3. In contrast, as the there is a significant reduction in RDP for Mobile IP as the two endpoints grow farther in

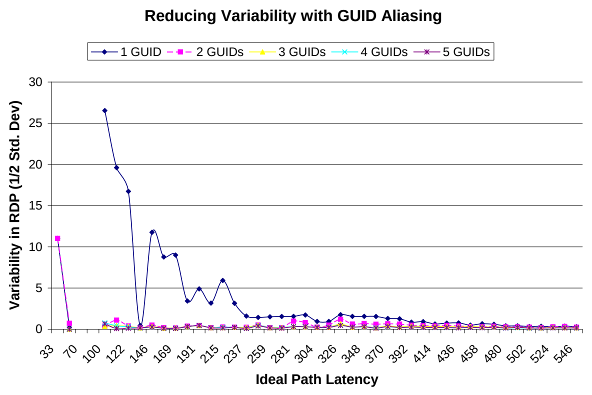


Figure 13: *Measuring the impact of additional routing roots* in Mobile Tapestry on RDP variability. We compare a single root to 2, 3, 4, and 5 roots.

distance. Mobile Tapestry's RDP, however, shows some initial variability in the local area, but then the RDP becomes relatively constant as the ideal path latency increases. Overall, Mobile Tapestry significantly outperforms Mobile IP under scenarios 1 and 3. Scenario 2 (MN close to or inside its home network) is the only instance where Mobile IP consistently outperforms Mobile Tapestry.

The Mobile Tapestry RDP results show significant variability, especially for low latency. The results are from simulations on 9 different topologies with 3 node assignments each (a total of 27 different experiments), and match those from [15], showing that overlay network performance depends on the specific topology and overlay layout.

In the following section, we show how the use of *GUID aliasing* eliminates this high variability, resulting in both low variance and low latency. We are also exploring the application of the techniques developed by the authors of [21] to reduce Mobile Tapestry's RDP under scenario 2 conditions.

5.3 Performance and Stability via Redundant Roots

In Section 3.4, we claimed that using GUID aliasing provides performance, performance variability, and fault-tolerance benefits. In this section, we present measurements that support that assertion.

First, we examine the impact of GUID aliasing on latency. Recall that for GUID aliasing, multiple GUIDs are generated deterministically for a mobile node by its proxy node, and advertised on its behalf. On the correspondent host's side, the Tapestry layer also calculates the same GUIDs, and multiplexes each message to all GUIDs in parallel. The mobile proxy node keeps a small cache of message identifiers, and discards duplicates once a message has been delivered to the mobile node. When used, GUID aliasing is transparent to both correspondent hosts and mobile nodes.

Effectively, message delivery latency under GUID aliasing is the delivery latency on the shortest forwarding path. Figure 12 shows that the latency RDP drops significantly as additional GUIDs are used. In particular, using 2 or more GUIDs yields an RDP value that is relatively stable and never more than 4 times the ideal path latency.

Next, we examine the impact of GUID aliasing on RDP variance. Figure 13 shows the $\frac{1}{2}$ standard deviation of the RDP as a function of the number of GUIDs. Using 2 GUIDs significantly decreases statistical variance across multiple topologies. The effect of multiple GUIDs is quite dramatic, given the high variance that we see in the single root experiments. We traced the high variance to a single topology scenario, where a very small number of Tapestry nodes are isolated on an remote branch of the network. The lack of nearby nodes causes all queries to route outside the local domain and, thus, incur high delay penalties even for relatively close objects. This example clearly shows how a particular overlay placement and physical topology combination can produce adverse results. However, as our results show, these effects can be reduced or eliminated by choosing the best path from two or more distinct paths.

These two experimental results demonstrate that GUID aliasing is a powerful tool that significantly improves latency and reduces performance variance, even when only 2 GUIDs are used. In Figure 14, we compare GUID aliasing with 2 and 5 GUIDs to two key Mobile IP scenarios. When the correspondent host is relatively close to the mobile node, Mobile Tapestry with GUID aliasing of 2 always outperforms Mobile IP.

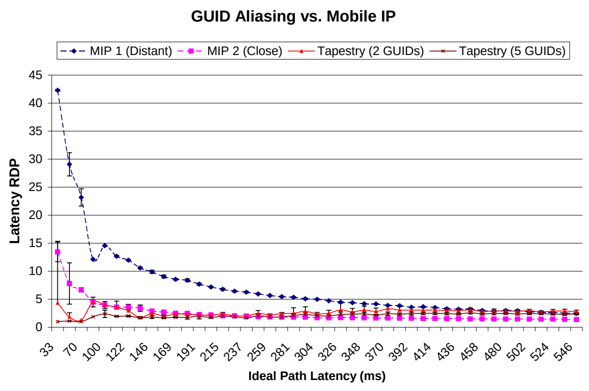


Figure 14: *Comparing the routing latency overhead* of Mobile IP in two scenarios (near to and far from the home network) to Mobile Tapestry with 2 and 5 GUIDs.

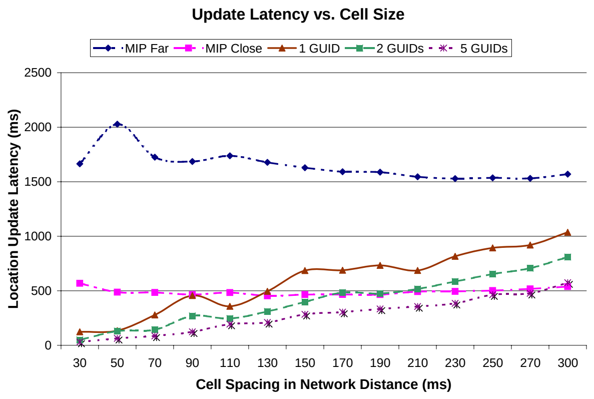


Figure 15: *Comparing the latency taken for a location update.* We compare the complete time required for location updates to settle, for Mobile IP in two scenarios, far from (1) and close to (2) the home network, and Mobile Tapestry with 1, 2 and 5 GUIDs. Latency to convergence is plotted versus cell spacing in network latency.

GUID aliasing imposes a higher load on the network, primarily in the form of increased bandwidth needs. However, given the ever increasing availability of high network bandwidth, we believe that GUID aliasing with 2 GUIDs is a beneficial tradeoff when low latency message delivery is required.

5.4 Rapid Mobility

We evaluate Mobile Tapestry's support for rapid mobility by examining several metrics, beginning with the latency to system convergence after a location update, for both Mobile IP and Mobile Tapestry. Location updates are generated when a node crosses a cell boundary, and convergence time is a function of the spacing between neighbor cells. We approximate the spacing between cells by the network distance between the foreign agents (proxy nodes).

As Figure 15 shows, in the case where the mobile node is roaming far from its home network, it can take between 1-2 seconds for confirmation to return after a location update. We note that this time is relatively constant regardless of the cell spacing, since the round trip time to and from the home agent dominates any measurement. In contrast, the time required to update all pointers in Mobile Tapestry is linear to the cell spacing. While cell spacing is highly variable depending on the wireless network, we expect spacing between neighbor cells in most networks to be a small number of ms (*i.e.*, $\leq$ 50ms, excluding inter-service provider domains). That means for these cell crossings, Mobile Tapestry takes $\leq$ 200ms to update all paths.

The metric above is different from the delay between the time a node crosses a cell boundary and the time the

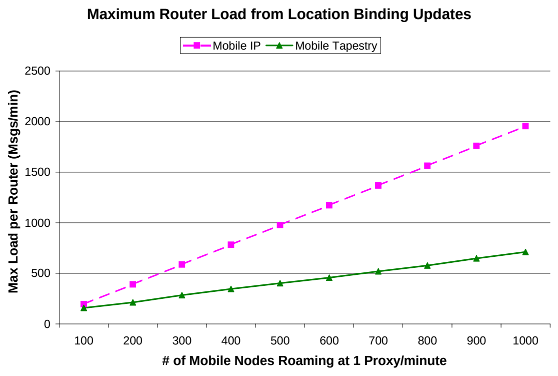


Figure 16: *Maximum Router Load from Location Binding Updates.* Assuming a single home agent for some home network, and a set of mobile nodes roaming and each requiring one location binding update per minute, we show maximum router load anywhere in the network for Mobile IP and Mobile Tapestry.

forwarding path can begin to route packets to the new proxy or foreign agent. This delay will be introduced into network traffic as jitter. For Mobile IP, the jitter is approximately half of the roundtrip time plotted above ($\approx$750ms on the wide-area). Mobile Tapestry, however, sets up a forwarding pointer from the old proxy to the new proxy as it registers with the new proxy. This means the delay introduced during cell crossing is the time taken for the new proxy to notify the old proxy, which is equal to the cell spacing metric in our graph. In most networks then, a Mobile Tapestry node incurs a jitter of less than 50ms during a cell crossing.

This jitter value is important when we consider the type of applications we can support while roaming. With jitter $\leq$ 50ms, Mobile Tapestry can support interactive applications even while rapidly crossing wireless cell boundaries. In contrast, Mobile IP's longer update time means applications could see serious performance degradation, beyond the acceptable limits of most multimedia applications.

A related property is the maximum movement rate that allows a connection to be maintained. The amount of time that a mobile node is available via an existing route is the time it stays connected to a single proxy, minus the time it takes to update its location binding. Using the results from Figure 15, it is clear that Mobile Tapestry's rapid location updates provide mobile nodes with a larger window in which to receive data. Furthermore, while Mobile IP assumes that packets in-flight to the previous proxy are lost and retransmitted by higher level protocols, Mobile Tapestry establishes a forwarding pointer between the proxies, ensuring that little data is lost in transit.

Finally, we examine the load placed on network routers by fast moving nodes. We make the following assumption: each home network contains a single home agent routing traffic for its mobile nodes, and each mobile node roams and requires one location binding update per minute. We then measure the maximum load placed on any router in the network by update messages as a function of number of mobile nodes. Figure 16 shows that as the number of mobile nodes increases, the load placed on routers in the network by Mobile Tapestry using a single GUID is less than half that of Mobile IP and grows at a lower linear rate. This result means that Mobile Tapestry can support more and faster moving nodes than Mobile IP, while using less bandwidth.

5.5 Fault-resilient Packet Delivery

Using simulation, we explored several factors relating to the fault-resilient packet delivery of messages from a correspondent host to a mobile node. First, since Mobile Tapestry exploits locality in choosing from multiple routing paths, messages in Mobile Tapestry, in general, travel shorter distances than those in Mobile IP. This distinction is shown in Figure 14. Since the probability of a failure in a system is proportional to the number of components, traversing a shorter path means a reduced likelihood of encountering failed component. Also, since there are multiple paths, Mobile Tapestry provides quick failover onto these backup routes.

Second, Mobile Tapestry routing uses short cut routing when a message intersects the forwarding path, reducing the overall path length. Finally, while traversing the forwarding path to the proxy, the presence of two-hop forwarding pointers means messages can avoid single link or node failures.

We are still conducting link and node failure experiments to gauge the benefits of these factors, however, our initial results show a minimum improvement of 40 percent increased availability for Mobile Tapestry relative to Mobile IP. Using multiple GUIDs further increases Mobile Tapestry's availability, although we have not yet fully quantified the exact amount of improvement.

5.6 Discussion

In this Section, we presented experiments that analyzed several aspects of the Mobile Tapestry system. A resonating theme through all of these measurements is that Mobile Tapestry achieves many of the goals of an ideal mobility system by leveraging locality in the underlying Tapestry object location layer. For example, Tapestry locality mechanisms remove the need for a home agent, and make routing and location update performance a function of the distance between communication endpoints, instead of the distance between a mobile node and its home agent. More specifically, Tapestry's locality mechanisms minimize message and location update latencies (Sections 5.2 and 5.4), reduce stress on network routers (Section 5.4), and reduce the likelihood of encountering failures in the network (Section 5.5). Additionally, we have shown that by using *GUID Aliasing* to trade bandwidth for performance , we can significantly reduce latency and variability.

Our experimental results also confirm the results reported by the authors of [15] and show that overlay networks, such as Mobile Tapestry, are more sensitive to changes in topology and overlay node placement than Mobile IP. However, by using GUID aliasing to trade bandwidth for performance, this variability can be largely removed. Overall, the experiments show that Mobile Tapestry offers significant advantages over Mobile IP, in terms of routing latency for normal messages and location updates, reduced stress on network routers, support for rapid mobility, and message delivery that is resilient to random faults in the network.

6 Related Work

Mobile IP is the widely accepted industry standard in mobility systems. As discussed in previous sections, Mobile Tapestry contains differences to Mobile IP in key design decisions, leading to dramatic improvements in performance, fault-tolerance, and ability to support rapid mobility. We do note that Mobile IP without route optimization supports better transparency because, unlike Mobile Tapestry, it does not require any modification to the correspondent host.

We also observe that GUID aliasing for cached routes in Mobile Tapestry is similar to route optimization in Mobile IP, in that both mechanisms need to be refreshed when a node moves to a new proxy. Unlike route optimization, however, an outdated cached forwarding path in Mobile Tapestry just performs like basic Mobile Tapestry; whereas an outdated route optimization cache results in a lost message, an application level timeout, and another triangle route through the home agent. An updated version of this paper will include comparison of GUID aliasing against route optimization.

There is also intensive ongoing research in the area of distributed overlays for location, including Chord [19], Content-addressable Networks (CAN) [12], and Pastry [17]. Like Tapestry, the Pastry system also uses a prefix-based hypercube routing scheme. Where Tapestry distributes location references to objects however, objects in Pastry are replicated and distributed by the system, making Pastry unsuitable for the type of indirection algorithms presented in this paper. The key difference between CAN, Chord and Tapestry is that unlike Tapestry, CAN and Chord do not consider network distances while building their overlay structures. Despite logarithmic logical hops to route to an object, each single logical hop can traverse the length of the network in these systems. While they use heuristics to improve performance, CAN and Chord do not preserve locality in object location. Therefore, application of Mobile Tapestry algorithms to CAN or Chord are likely to produce much less efficient results.

Finally, the Internet Indirection Infrastructure [18] project uses location systems such as Chord to embed a level of indirection in the network, supporting services such as multicast, anycast and mobile nodes. The main difference between a mobile system built on *I3* and Mobile Tapestry is that, while the former provides a single level of indirection as a rendezvous point and for redirecting packets, Mobile Tapestry provides multiple points of indirection, one at every overlay hop from the "root node" of the mobile node to the proxy node, where each indirection point can perform *I3*'s redirection functionality. Furthermore, these points of indirection are self-organized in order to exploit locality and minimize latency in location updates and forwarding packets. Finally, Mobile Tapestry utilizes the available type indirection in Tapestry to build an extendable hierarchy of mobility support (Section 4.4).

7 Conclusion

This paper explores the design and implementation of *Mobile Tapestry*, a set of mobility extensions to the Tapestry overlay network. Basic Tapestry provides efficient, locality-biased routing of queries to objects anywhere in the infrastructure. Mobile Tapestry leverages this capability to route messages to mobile nodes and groups of nodes.

With the addition of mechanisms for rapid mobility, path selection, and security, Mobile Tapestry provides many advantages over Mobile IP: it incurs lower routing latency, provides faster adaptation to change and failure, and causes less stress on the network. As a fringe benefit, mobile nodes inherit the ability of basic Tapestry nodes to publish objects so that others can route to them quickly. We are implementing Mobile Tapestry as a part of the OceanStore [14] distributed information repository project. When completed, Mobile Tapestry will provide a powerful tool for mobile access to distributed information.

References

[1] Balakrishnan, H., Seshan, S., and Katz, R. H. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks 1*, 4 (December 1995).

[2] Boggs, D. R. *Internet Broadcasting*. PhD thesis, Xerox PARC, October 1983. Available as Technical Report CSL-83-3.

[3] Chu, Y., Rao, S. G., and Zhang, H. A case for end system multicast. In *Proceedings of SIGMETRICS* (June 2000), ACM, pp. 1–12.

[4] Douceur, J. R. The Sybil attack. In *Proceedings of IPTPS* (March 2002).

[5] Droms, R. *Dynamic Host Configuration Protocol*, Oct. 1993.

[6] 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.

[7] Johnson, D. B. Scalable support for transparent mobile host internetworking. *Wireless Networks 1*, 3 (October 1995), 311–321. special issue on "Recent Advances in Wireless Networking Technology".

[8] Johnson, D. B., and Perkins, C. Mobility support in IPv6, July 2001. `http://www.ietf.org/internet-drafts/draft-ietf-mobileip-ipv6-15.txt`.

[9] Myles, A., Johnson, D. B., and Perkins, C. A mobile host protocol supporting route optimization and authentication. *IEEE Journal on Selected Areas in Communications 13*, 5 (June 1995), 839–849.

[10] Perkins, C. IP Mobility Support. IETF, October 1996. RFC 2002.

[11] 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.

[12] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A scalable content-addressable network. In *Proceedings of SIGCOMM* (August 2001).

[13] Rekhter, Y., and Li, T. An architecture for IP address allocation with CIDR. RFC 1518, `http://www.isi.edu/in-notes/rfc1518.txt`, 1993.

[14] Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., and Kubiatowicz, J. Pond: The first OceanStore prototype. In *Proceedings of FAST* (San Francisco, April 2003), USENIX.

[15] Rhea, S. C., and Kubiatowicz, J. Probabilistic location and routing. In *Proceedings of INFOCOM* (To appear in June 2002).

[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] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. Internet indirection infrastructure. In *Proceedings of SIGCOMM* (August 2002), ACM.

[19] 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).

[20] Zegura, E. W., Calvert, K., and Bhattacharjee, S. How to model an internetwork. In *Proceedings of IEEE INFOCOM* (1996).

[21] 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).

[22] 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.