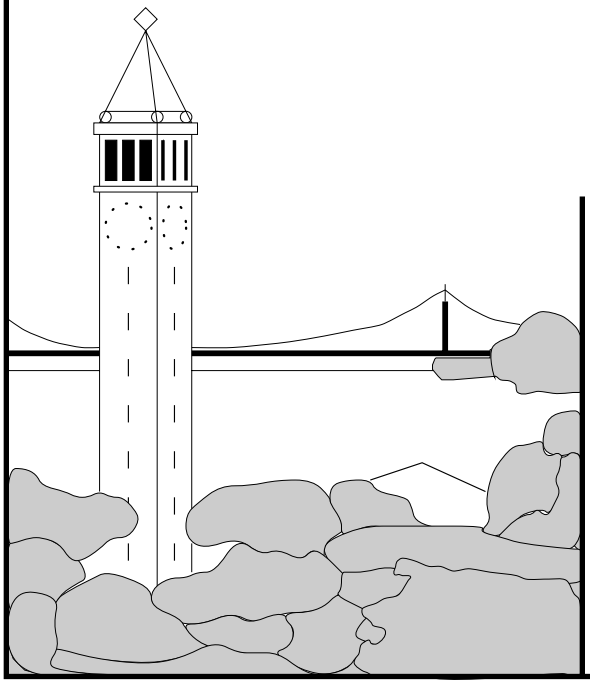
Host Mobility Using an Internet Indirection Infrastructure

Shelley Zhuang *Kevin Lai* *Ion Stoica* *Randy Katz*
Scott Shenker
{*shelleyz, laik, istoica, randy*}*@eecs.berkeley.edu, shenker@icsi.berkeley.edu*
CS Division, EECS Department, U.C.Berkeley



Report No. UCB/CSD-2-1186

June 2002

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

This technical report is supported by grant number DABT63-98-C-0038

Host Mobility Using an Internet Indirection Infrastructure

Shelley Zhuang Kevin Lai Ion Stoica Randy Katz
Scott Shenker
{shelleyz, laik, istoica, randy}@eecs.berkeley.edu, shenker@icsi.berkeley.edu
CS Division, EECS Department, U.C.Berkeley

June 2002

Abstract

We propose the Robust Overlay Architecture for Mobility (ROAM) to provide seamless mobility for Internet hosts. This architecture uses an indirection infrastructure that provides a rendezvous communication abstraction: instead of explicitly sending packets to a destination address, packets are sent to an identifier. A receiver who wishes to receive those packets uses the indirection infrastructure to associate its address with the identifier.

ROAM allows end-hosts to avoid the inefficiency of triangle routing by choosing nearby indirection points, and it is as robust as the underlying IP network to node failure. In addition, it preserves location privacy and allows end hosts to move simultaneously. We have developed a user-level prototype system on Linux that provides transparent mobility without modifying applications or the TCP/IP protocol stack. We also present both simulation and experimental results.

1 Introduction

While the wired Internet reaches many homes and businesses, the wireless Internet has the potential to not just reach, but encompass all the spaces that people use to live, work, and travel. Wireless data services (e.g., 802.11b, GPRS, 3G cellular) with differing bandwidth, latency, cost, and coverage will soon provide the potential for seamless, though heterogeneous, coverage. Users will also use wireless connectivity differently. Applications that are easier to use while mobile (e.g., IP telephony, instant messaging, and audio streaming) will be as popular as email and web browsing. In this environment, people will want both seamless connectivity (flows uninterrupted by mobility) and continuous reachability (the ability of other hosts to contact the user's host despite mobility).

Unfortunately, the standard Internet cannot provide these services. The fundamental problem is that the Internet uses IP addresses to combine the notion of unique host identifier with location in the network topology. For a mobile host to have seamless connectivity and continuous reachability, it must retain its identifier while changing its location. Previous mobility proposals decouple this binding by introducing a fixed indirection point (e.g., Mobile IP [20]), redirecting through the Domain Name System (e.g., TCP Migrate [24]), or using indirection at the link layer (e.g., cellular mobility schemes).

However, these proposals lack one or more of the following properties that are highly desirable in the environment described above:

- **Efficient routing** depends on routing packets on paths with latency close to the shortest path provided by IP routing. This is important for delay sensitive applications like IP telephony.
- **Fault tolerance** requires surviving the failure or overload of hosts and links. Communication between mobile hosts should not be more vulnerable to faults than communication between stationary hosts.
- **Preservation of location privacy** requires minimizing the number of hosts trusted with knowing the mobile host's network topological location, assuming it correlates with geographical location. Mobile users will usually not be anonymous with respect to their correspondent user when using applications like IP telephony and therefore may not wish to expose their geographic location.
- **Simultaneous mobility** is the simultaneous movement of both end points during or immediately prior to a session. For person-to-person applications like IP telephony, this is likely to happen at least occasionally.
- **Link layer independence** allows operation across heterogeneous link layer technologies, not all of which support the same link layer mobility scheme (e.g., GSM mobility).
- **Personal/session mobility** allows a user to redirect a new session or migrate an active one from one application or device to another when a better choice becomes available [1] [16] [30]. Applications or devices may fail (e.g., a cell phone may have poor coverage indoors) or a better performance/price option may become available (e.g., video conferencing on a laptop connected to a free 11Mb/s 802.11b network is preferable to a cell phone connected to a 128Kb/s GPRS network).

To provide these properties, we propose the Robust Overlay Architecture for Mobility (ROAM). Our approach is to use the Internet Indirection Infrastructure ($i3$) [27]. $i3$ is implemented as an overlay network on top of the existing IP network, and provides a *rendezvous*-based communication abstraction. In $i3$, each packet is sent to an identifier. To receive a packet, a receiver inserts a trigger, which is an association between the packet's identifier and the receiver's address. The trigger is stored at a server in the $i3$ overlay network. Each packet is routed through the overlay network until it reaches the $i3$ server which stores the trigger. Once the matching trigger is found the packet is forwarded to the address specified by the trigger. Thus, the trigger plays the role of an *indirection* point that relays packets from the sender to the receiver.

ROAM addresses each of the properties described above. To provide efficient routing, receivers can choose trigger identifiers that map to nearby servers. Senders can avoid the overhead of overlay network routing by caching the addresses of trigger servers. Because the overlay network is inherently robust against failures, $i3$ preserves connectivity in the presence of multiple overlay node failures. Our solution achieves location privacy because it provides end host identifiers independent of end hosts' network attachment points, which reveal location. Only knowledge of the the trigger identifier is required to communicate. ROAM provides simultaneous mobility because the $i3$ network serves as an anchor point for the two communication end points. Unlike GSM mobility, ROAM is not specific to a particular network technology and allows roaming to any network that delivers IP packets. Since an $i3$ identifier can be bound to a host, session, or person (unlike Mobile IP, where an IP address can only be bound to a host), personal/session mobility applications can leverage the ROAM infrastructure for efficiency, fault tolerance, and privacy. Leveraging a single ROAM infrastructure across link layer technologies and mobility models (network vs. personal/session) significantly reduces the total cost of deployment.

This paper makes several contributions. First, it demonstrates the natural fit between a rendezvous-based communication abstraction and mobility. Second, it demonstrates the benefits of a mobility architecture based on a shared overlay network. Such a solution leverages the robustness and efficient routing of overlay networks. Finally, it proposes an efficient and robust IP mobility solution that does not require changes of either the applications running on mobile hosts or the operating system.

We simulate ROAM and Mobile IP using a variety of network topologies, host mobility models, and communication models. We show that as the number of servers increases in a transit-stub network, ROAM's latency decreases until it reaches 0.25-40% that of Mobile IP, depending on the mobility and communication model. In addition, ROAM is as robust to fail-stop faults as the underlying IP network, while Mobile IP fails to provide connectivity to 50% of communication pairs when only 15% of network nodes fail.

We use a proxy based solution to transparently support unmodified applications on an unmodified Linux kernel. Using our prototype implementation, we show that our solution can perform rapid soft handoffs with no noticeable disruption of TCP throughput.

The paper is organized as follows. We review related work in Section 2, and provide an overview of $i3$ in Section 3. In Section 4 we discuss the design of our ROAM solution, and in Section 5 we present some implementation details. In Section 6 we present the results of our simulation and implementation performance experiments to evaluate ROAM. We discuss some open issues in Section 7, and conclude with a summary in Section 8.

2 Related Work

In this section we review the main mobility proposals.

Several link layer technologies provide mobility at the link layer (e.g., as in Ricochet [22], 802.11b, or GSM). However, these solutions preclude mobility across link layer technologies. In addition, hiding mobility at the link layer results in a reinvention of mobility support in each new wireless system; solving the mobility problem at the network layer results in a reusable mobility infrastructure for all link technologies.

One proposal to achieve mobility in the Internet is Mobile IP (MIP). MIP in IPv4 [20] and IPv6 [13] uses an explicit indirection point, called the Home Agent (HA), to encapsulate and relay the Correspondent Host's (CH) initial packet to the mobile host (MH). MIP provides the following options that determine how the following packets are routed: 1) triangle routing, 2) bidirectional tunneling, and 3) route optimization.

As noted by Cheshire and Baker [5] no MIP routing option is clearly better than the others; instead, different options are suitable for different circumstances. Options (1) and (2) preserve location privacy, but routing can be inefficient when the MH and CH are close relative to their distance from the HA. With route optimization (an extension in MIPv4 [21], but standard in MIPv6), the MH conveys its care-of IP address to the CH using a Binding Update (BU). Routing is efficient because the ratio of the latency of the optimized route to the latency of the shortest IP path (or *stretch*) is 1.0. However, the CH must be modified to support MIPv4 with route optimization or IPv6. This also exposes the MH's current care-of address (and therefore its location) to the CH, thus compromising location privacy. In certain delay-sensitive or real-time applications, the latency involved in handoffs can be above the threshold if the MH is far away from the CH.

In general, the dependence in MIP on a fixed HA reduces fault tolerance. If the HA or its network fails or is overloaded, then the MH will be unreachable.

In order to address routing anomalies and robustness issues associated with a fixed HA, researchers are starting to propose the notion of dynamic home agents in MIPv4 [4]. However, the actual algorithm used to discover and allocate a nearby home agent is still under investigation. MIPv6 provides a *dynamic home agent address discovery* mechanism [13] that allows a MH to dynamically discover the IP address of a HA on its home network. This scheme increases the robustness of MIPv6 as the HA is no longer a statically fixed entity, but it does not address routing inefficiencies caused by routing through the HA when the MH is far away from its home network.

Supporting Mobility for TCP with SIP [29] spoofs constant TCP endpoints in a similar way to MIP with route optimization. Realization of this requires modifying the IP stack of the CH.

The Mobility Support using Multicasting in IP (MSM-IP) architecture [17, 18] implements mobility using IP Multicast [7]. The main advantages of MSM-IP are that it can have low routing stretch and do handoffs with little or no packet loss. Several studies [18] [11] [12] have shown that multicast mobility can cut the routing stretch of Mobile IP in half and significantly reduce packet loss due to handoffs. However, the MSM-IP location service is a single point of failure and is vulnerable to overload, network faults, and host faults.

In TCP Migrate [24], both the MH and CH use a modified form of TCP which can tolerate a change in IP address during a connection. The CH uses DNS to learn the current address of the MH, who updates DNS every time it moves. Since TCP Migrate does not use an indirection point, it can achieve an optimal latency stretch of 1.0 and is as fault tolerant as IP routing. On the downside, it lacks simultaneous mobility support, requires modification of the TCP implementations on both the MH and the CH, and does not preserve location privacy. TCP Migrate is well suited for person-to-server applications with short-lived flows like email and web browsing.

The mobility schemes previously described in this section track mobile hosts. In contrast, personal and session mobility schemes (e.g., The Mobile People Architecture (MPA) [16] ICEBERG [30], and Telephony Over Packet networkS [1]) track people or sessions. This allows redirection of new sessions or migration of active sessions to a completely different application or device according to user connectivity (e.g., which devices are currently accessible to the user) and user preferences (e.g., less expensive or higher performance). In contrast, Mobile IP redirects flows to the same device regardless of whether the user can actually use the device (e.g., the user may have left it somewhere or it may not have power or connectivity). The costs of personal/session mobility schemes are modifications to applications (unlike Mobile IP) and an indirection infrastructure (e.g., the Personal Proxy in MPA).

$i3$'s Application Programming Interface (API)	
$sendPacket(p)$	send packet
$insertTrigger(t)$	insert trigger
$removeTrigger(t)$	remove trigger

(a)

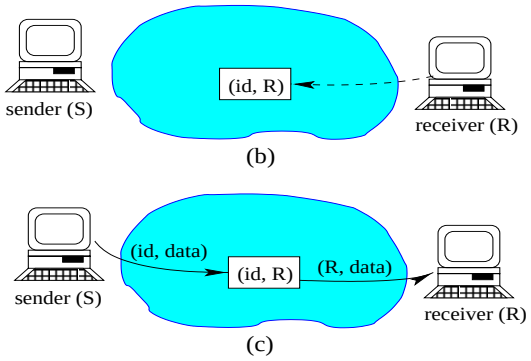


Figure 1: (a) $i3$'s API. Example illustrating communication between two nodes: (b) The receiver R inserts trigger (id, R). (c) The sender sends packet $(id, data)$.

In contrast to all of the above schemes, the novelty of our approach is the use of an overlay infrastructure that supports a rendezvous-based communication abstraction. As a result, ROAM is able to achieve efficiency, robustness, location privacy, and simultaneous mobility. In addition, the flexibility of $i3$ identifiers allows ROAM to support mobility at any layer. $i3$ identifiers can be bound to hosts (the focus of our design in Section 4, as in Mobile IP, so ROAM can support mobility without modifications to applications. $i3$ identifiers can also be bound to sessions and people (discussed briefly in Section 7), as in ICEBERG and the Mobile People Architecture, thus serving as an enhanced infrastructure for personal/session mobility-aware applications.

The cost of using ROAM is the deployment of $i3$ servers in the Internet and ROAM proxies on both the mobile and correspondent hosts. We quantify the trade off between infrastructure deployment and routing efficiency in Section 6 and describe operation without the CH's proxy in Section 7.

3 Background

In this section we present a brief overview of an Internet indirection infrastructure, $i3$ [27], which forms the foundation for our mobility solution. The purpose of $i3$ is to provide indirection; that is, it decouples the act of sending from the act of receiving. The $i3$ service model is simple: sources send packets to a logical *identifier*, and receivers express interest in packets sent to an identifier. Delivery is best-effort like in today's Internet, with no guarantees about packet delivery.

3.1 Rendezvous-based Communication

The service model is instantiated as a rendezvous-based communication abstraction. In their simplest form, packets are pairs $(id, data)$ where id is an m-bit identifier[1] and $data$ is the payload (typically a normal IP packet payload). Receivers use *triggers* to indicate their interest in packets. In their simplest form, triggers are pairs $(id, addr)$, where id is the trigger identifier, and $addr$ is a node's address, consisting of an IP address and UDP port number. A trigger $(id, addr)$ indicates that all packets sent to identifier id should be forwarded (at the IP layer) by the $i3$ infrastructure to the node with address $addr$. More specifically, the rendezvous-based communication abstraction exports the three primitives shown in Figure 1(a).

Figure 1(b) illustrates the communication between two nodes, where receiver R wants to receive packets sent to id. The receiver inserts the trigger (id, R) into the network. When a packet is sent to identifier id, the trigger causes it to be forwarded via IP to R.

The above description was of the simplest form of the abstraction. $i3$ also provides a generalization that allows inexact matching between identifiers. We assume identifiers are m bits and that there is some *exact-match threshold* k with $k < m$. Thus, a trigger identifier id_t matches a packet identifier id if and only if id_t is a longest prefix match (among all other trigger identifiers) and this prefix match is at least as long as the exact-match threshold k. The value k is chosen to be large enough so that the probability that two randomly chosen identifiers match is negligible. [2]

Thus, as in IP multicast, the identifier id represents a logical rendezvous between the sender's packets and the receiver's trigger. This level of indirection decouples the sender from the receiver and enables them to be oblivious to the other's location. However, unlike IP multicast, hosts in $i3$ are free to place their triggers. This can alleviate the triangle routing problem in Mobile IP. In addition, $i3$ can be generalized to support multicast, anycast, and service composition. For more details refer to [27].

3.2 $i3$ Implementation

$i3$ is implemented as an overlay network which consists of a set of servers that store triggers and forward packets (using IP) between $i3$ nodes and to end hosts.

To maintain this overlay network and to route packets in $i3$, we use the Chord lookup protocol [6]. Chord assumes a circular identifier space of integers $[0, 2^m)$, where 0 follows $2^m - 1$. Every $i3$ server has an identifier in this space, and all trigger identifiers belong to the same identifier space. The $i3$ server with identifier n is responsible for all identifiers in the interval $(n_p, n]$, where n_p is the identifier of the node preceding n on the identifier circle. Figure 2(a) shows an identifier circle for $m = 6$. There are five $i3$ servers in the system with identifiers 5, 16, 24, 36, and 50, respectively. All identifiers in the range (5, 16] are mapped on server 16, identifiers in (17, 24] are mapped on server 24, and so on.

When a trigger $(id, addr)$ is inserted, it is stored on the $i3$ node responsible for id. When a packet is sent to id it is routed by $i3$ to the node responsible for its id; there it is matched against (any) triggers for that id and forwarded (using IP) to all hosts interested in packets sent to that identifier. Chord ensures that the server responsible for a identifier is found after visiting at most $O(\log n)$ other $i3$ servers irrespective of the starting server (n represents the total number of servers in the system). To achieve this, Chord requires each node to maintain only $O(\log n)$ routing state. Chord allows servers to leave and join dynamically, and it is highly robust against failures. For more details refer to [6]. Figure 2(b) shows an example in which trigger $(30, R)$ is inserted at node 36 (i.e., the node that maps $(24, 36]$, and thus is responsible for identifier 30). Packet $(30, data)$ is forwarded to server 30, matched against trigger $(30, R)$, and then forwarded via IP to R.

Note that packets are not stored in $i3$; they are only forwarded. End hosts use periodic refreshing to maintain their triggers in $i3$. Hosts need only know one $i3$ node to use the $i3$ infrastructure. This can be done through a static configuration file, or by a DNS lookup assuming $i3$ is associated with a DNS domain name. In Figure 2(b), the sender knows only server 16, and the receiver knows only server 5.

4 ROAM Design

In this section, we describe ROAM, which provides an end-to-end architecture for Internet host mobility using the $i3$ network as a level of indirection.

Achieving host mobility on top of the basic $i3$ architecture is straightforward. A mobile host that changes its address from R to R' as a result of moving from one subnetwork to another can preserve end-to-end connectivity by simply updating each of its existing triggers from (id, R) to (id, R'), as shown in Figure 3. ROAM exhibits the following desirable properties:

Efficiency:

- **Efficient routing:** By caching trigger server addresses and carefully selecting low latency trigger servers as described below, we can reduce the stretch from using ROAM to below 1.5 (see Section 6.1).

[1] In the implementation presented in this paper, we use $m = 256$. Such a large value of m allows end hosts to choose trigger identifiers independently since the chance of collision is minimal. In addition, a large m makes it very hard for an attacker to guess a particular trigger identifier.

[2] In our implementation we choose $m = 256$ and $k = 128$, and triggers with the same k-bit prefix are stored on the same $i3$ server.

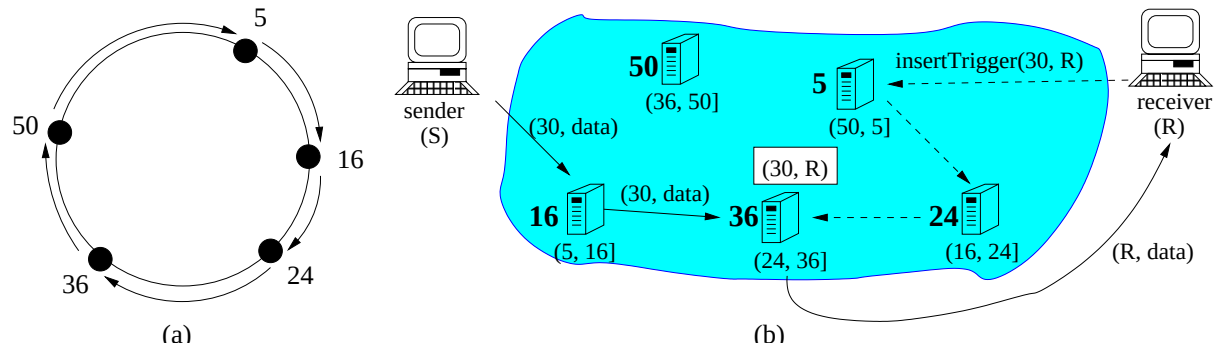


Figure 2: (a) A Chord identifier circle for $m = 6$, with 5 servers identified by 5, 16, 24, 36, and 50, respectively. Each server is responsible for all identifiers between its identifier and the identifier of the node that precedes it on the circle. (b) Receiver R inserts trigger $(30, R)$, and the trigger is forwarded via $i3$ to server 36 which is responsible for identifier 30. The trigger is stored there (shown in the white box) until explicitly removed or timed out. When sender S sends packet $(30, data)$, it is also forwarded via $i3$ to server 36. Servers identifiers are in bold. The interval of identifiers for which each server is responsible are also shown.

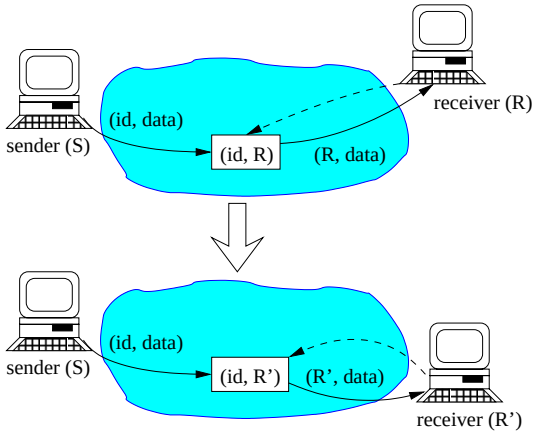


Figure 3: Upon changing its address from R to R', a receiver needs only to update its trigger. This change is transparent to the sender. Also, any change of the sender's location is transparent to the receiver.

- **Fast handoff:** Since the MH sends trigger updates to its nearby trigger server, it can perform lower latency handoffs than mobility systems such as Mobile IP or TCP Migrate.

Robustness:

- **Infrastructure:** Since triggers are periodically refreshed, ROAM recovers gracefully from server failure. If a server fails, the triggers stored at that server are inserted at another server the next time they are refreshed. To make $i3$ server failure completely transparent to end-hosts, one possibility is to have $i3$ replicate triggers. For more details refer to [27].
- **Multicast-based soft handoff:** A MH can set triggers pointing to two network addresses that it may have access to during handoff, and use $i3$ multicast to receive packets on both addresses, thereby eliminating packet loss.

Privacy:

- **Location privacy:** The sending host need not be aware of the MH's current IP address, thus preserving location privacy. Since two hosts need only know the other's trigger to communicate, and triggers can be chosen to not reveal any location information, the exact location of the end hosts can be completely hidden.[3]
- **Restricted eavesdropping:** By using both public and private triggers (described in Section 4.4), we can limit the possibility for eavesdropping in ROAM to the same as in the Internet.

Simultaneous mobility: The sending and receiving hosts can move simultaneously while the $i3$ network serves as an anchor point for the two sides of the communication channel.

In the following sections, we describe efficient routing, fast handoff, multicast-based soft handoff, and restricted eavesdropping in more detail. We then discuss how ROAM supports native and legacy applications.

4.1 Efficient Routing

Although the Chord lookup protocol limits the number of hops traversed in the $i3$ overlay network to $O(\log n)$, the delay on each hop may be comparable or even larger than the IP shortest path between the MH and the CH. This can result in unacceptably high delay. We reduce this delay by using the following techniques: 1) end hosts cache the mapping from triggers to $i3$ server IP addresses, 2) end hosts sample the delay to random $i3$ servers to find a close one, and 3) a mobile

[3]However, choosing a trigger that is far away may have a negative impact on routing efficiency.

host (MH) uses *mobility-aware* caching of past close triggers to increase the effectiveness and lower the cost of sampling.

4.1.1 Caching the Trigger-to-Address Mapping

Although an end host must traverse $O(\log n)$ hops initially to lookup a trigger, it can then cache the mapping of that trigger to the IP address of the $i3$ server which stores that trigger. As long as that server continues to store that trigger, the end host can then send trigger refreshes and packets with the same identifiers directly to that server. For example, in Figure 2, both the sender (S) and the receiver (R) would cache server 36. Since the trigger can be reused across flows and applications, the $O(\log n)$ traversal only needs to be done when $i3$ servers fail or when using a trigger for the first time (e.g. when sampling the delay for new triggers as described below).

4.1.2 Sampling

While the above scheme ensures that subsequent packets will only have to traverse one hop in the overlay network, the delay can still be large if the $i3$ server storing the matching trigger is far from both end-hosts. To address this, after moving to a new location, an end host picks triggers with random identifiers and then measures the round trip delay to the servers that store those triggers. The end host then simply uses the trigger with the lowest delay. The total amount of data required for setting a trigger, acknowledging the setting of a trigger, sending an empty packet to a trigger, and receiving that empty packet is an average of 4 * 45.8 bytes * 8 bits/byte = 1466 bits (assuming header compression, see Section 7). Even for a system with $16{,}384$ $i3$ servers, only 32 samples results in a 90th percentile latency stretch of 1.5 [27]. A MH taking 32 samples at a rate of once per second uses 1.466 Kb/s of bandwidth for 32 seconds after each move. This probing can occur in parallel with the MH's other communication because end-hosts can dynamically change triggers without disrupting end-to-end connectivity.

4.1.3 Mobility-Aware Trigger Caching

We can further optimize routing by using *mobility-aware* caching of past close triggers to increase the effectiveness and lower the cost of the sampling. An alternative to mobility-aware caching would be to increase the number of samples taken, which would further reduce the stretch. However, this would also increase the overhead. Instead, by caching past close triggers which are likely to be close again, we gain the benefit of having sampled more triggers without the cost.

We assume that mobile hosts are likely to move in a pattern where most moves are short (in geographic distance and network latency), but some moves are very far [28]. This pattern corresponds to a person who drives around a metropolitan area which is a few 10's of miles in diameter, but occasionally flies hundreds or thousands of miles to another location.

We cache sampled triggers to take advantage of this pattern. The goal is to create diversity in the cache so that a trigger in the cache is near each of the remote locations that a mobile host visits (perhaps infrequently), while preventing the frequent local moves from polluting the cache. When the mobile host changes its network address, it randomly samples $i3$ servers as described above, caches the result, and measures the delay to every trigger in the cache. When the cache is full, and the new sample is closer than any in the cache, then we must select a cache entry to evict. If the new sample is much closer than the next closest cache entry (e.g., the new sample's latency is less than 50% of the latency of lowest latency cache entry), then we replace the least recently used trigger in the cache. That the new sample is much closer than the next closest sample indicates that the mobile host is probably at a location that is far from any it has visited before, so we evict the entry we are least likely to use again. If instead the new sample is not much closer than the next closest entry in the cache is (e.g. the new sample's latency is 50%-100% of the latency of the next closest cached trigger), then we replace that entry with the new sample. This indicates that the mobile host is relatively close to a recently visited location, and the new sample is a better server for that location.

Using the mobility aware caching algorithm, a mobile host taking 32 samples for each move, caching 10 entries, and using a 50% threshold as described above reduces the latency stretch to nearly 1.0 for a sufficient number of $i3$ servers (see Section 6.1).

4.1.4 Privacy-Efficiency Tradeoff

Although the location of a nearby i3 server reveals the MH's location to some extent, the MH can choose a private trigger x such that it is stored on the i3 server close to the CH. This also results in a low latency stretch without compromising location privacy. With the current implementation this can be achieved by having x share the same k-bit prefix as the private trigger of the CH. For even more location privacy, both end-points can choose completely random $i3$ servers. The flexibility of $i3$ allows each application to make this tradeoff as desired.

4.2 Fast Handoff

Existing mobility systems such as Mobile IP or TCP Migrate propagate address binding updates (BUs) all the way to a HA or CH. As a result, a potentially large number of packets may be in flight when the path from the MH to the HA or CH is long. If the MH stops receiving packets at the old IP address before starting to receive packets at the new address (*cold switching*), then those in-flight packets will be lost.

Low latency handoff [15] and fast handover [31] extensions to MIPv4 and MIPv6 propose two mechanisms to address

this problem. The first mechanism attempts to send a BU in advance of an actual link-layer handoff when the handoff is anticipated. However, timing must be arranged such that the BU completes before the actual handoff does, which may be hard to achieve in practice. Similar in concept to Regionalized Tunnel Management [10] and Hierarchical Mobility [26] extensions in MIPv4 and MIPv6, the second mechanism sets up a bi-directional tunnel between an anchor Foreign Agent (FA) that stays the same during rapid movements and the current FA. This allows the MH to delay a formal BU to the HA which minimizes the impact on real-time applications. However, this mechanism relies on the existence of a FA in *each* network the MH visits. Furthermore, the use of link-layer triggers and inter-FA advertisements in these mechanisms assumes a homegenous link-layer technology.

In contrast, ROAM provides a simple mechanism whereby receivers can choose indirection points (i.e., triggers) that map onto nearby $i3$ servers. Since the number of packets that are lost during a cold-switch is proportional to the delay between the MH and the indirection point, we expect that the ability of a MH to choose nearby triggers will significantly reduce packet loss. In section 6.2.3, we compare how well ROAM supports cold-switches with MIPv6 via experiments.

4.3 Multicast-based Soft Handoff

When a MH moves from one network to another, there may be an interval during which it has poor connectivity (either lost packets or low bandwidth) in the new network, but good connectivity in the old network. If the MH performs handoff too early, then its performance can suffer from poor connectivity in the new network. On the other hand, if the MH performs handoff too late, then it may lose packets as the connectivity in the old network degrades.

The solution in ROAM is to use the generalized level of indirection provided by $i3$ to do multicast-based soft handoff. In the situation described above, when the MH can obtain an address in the new network, the MH's proxy inserts a trigger with the same identifier as its existing trigger, but associated with the new address. This causes the same packets to be delivered to both the old and new addresses. This allows the MH to take advantage of the best available connectivity. When the connectivity in the new network is poor and connectivity in the old network is good, most packets will arrive via the old address. When the reverse is true, most packets will arrive via the new address. We address the problems of determining when to stop using multicast and how to suppress duplicate packets in Section 5.1.

4.4 Restrict Eavesdropping: Public and Private Triggers

The original $i3$ design supports multicast by allowing any host in the network to add a trigger with the same identifier as another host's trigger. However, this allows any host to eavesdrop on another host's communications *if* it knows that host's trigger.

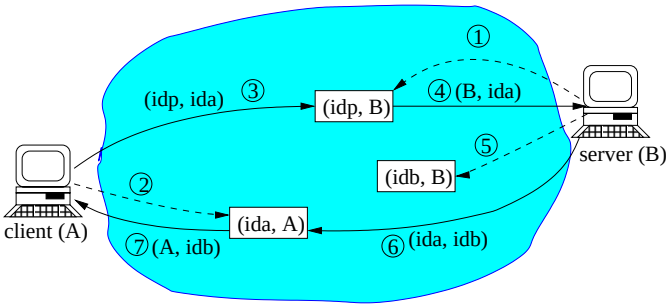


Figure 4: Example of setting up a connection via private triggers id_a and id_b between client A and server B. id_p represents B's public trigger.

We solve this by differentiating between *public* and *private* triggers. Private triggers are secretly chosen by the application end-points. Public triggers can be computed by all end-hosts in the system and are used to establish initial contact with the desired end-host. For example, the "New York Times" web server would have a public trigger that is a hash of its name. Subsequently, the client and the web server use the public trigger to choose a pair of private triggers, and then use these private triggers to exchange the actual data.

To avoid eavesdropping on public triggers, users can set the EXCLUSIVE_ID flag in the trigger headers to preclude other hosts from inserting triggers with the same identifier. Since private triggers are assumed to be secret, they don't need to have the EXCLUSIVE_ID flag set. This allows applications to use the multicast functionality via private triggers (see Section 4.3). Also, end hosts are free to choose their private triggers such that they are stored on nearby servers.

Section 7 discusses another possible attack in which a malicious user may hijack a host's public trigger.

4.5 Support for Native Applications

Figure 4 shows the details of our scheme for an $i3$ native application. Consider the application where a client A accesses a web server B. The web server B maintains a public trigger with identifier id_p in $i3$ (step 1). The control path operations are as follows. Client A inserts a private trigger with identifier id_a into $i3$ (step 2), and sends id_a to web server B via B's public trigger id_p (step 3). B receives id_a from $i3$ (step 4) and inserts a private trigger with identifier id_b into $i3$ (step 5). B then sends id_b to A via A's private trigger id_a (step 6), and A receives it from $i3$ (step 7). Finally, data packets from A to B flow through B's private trigger id_b, and through A's private trigger id_a in the reverse direction.

Notation	Definition
$X.hip$	home IP address of host X
$X.cip$	current IP address of host X
$C.port$	port associated to client process C
$i3\_hdr$	$i3$ packet header (see Figure 6)
$i3\_hdr.id$	$i3$ packet's identifier
$proxy\_hdr$	$i3$ proxy header
$proxy\_hdr.$ $flags$	flags: ID_MASK specifies that $proxy\_hdr$ has an $i3$ identifier. DATA_MASK specifies that the payload has an IP packet.
$H()$	well-known hash function; used to compute public trigger identifier for X as $H(X.hip)$
$trans\_table$	translation table maintained by each $i3$ proxy; each entry is a pair (IP address, $i3$ identifier)

Table 1: Notations used in Section 4.6.

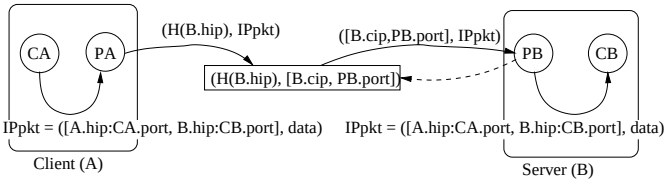


Figure 5: Supporting legacy applications. $A.hip$ and $B.hip$ represents home IP addresses of hosts A, and B, respectively. Each host has a proxy that intercepts applications packets and send them via $i3$.

4.6 Support for Legacy Applications

Although achieving host mobility for $i3$ native applications is straight-forward, many legacy applications will remain $i3$ unaware. In designing a solution for these applications, our primary goals are to remain transparent to both applications and the TCP/IP protocol stack. The main host modification required for legacy applications is a user-level ROAM proxy. The proxy serves the following functions: (1) encapsulates and decapsulates IP packets within $i3$ packets, (2) determines the triggers of remote hosts, and (3) sends the local private trigger to remote hosts. Table 1 gives the notations used in this section.

We assume that each host X has a current IP address denoted by $X.cip$ and a home IP address (e.g., the address of the host in its home network) denoted by $X.hip$. The home address is stored in the end-host's DNS record, and it is used as a source address for all packets sent by legacy applications

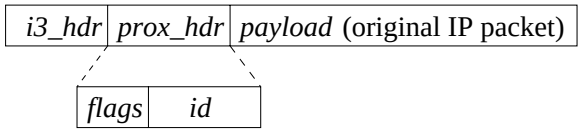


Figure 6: The format of the $i3$ packet handled by the proxy. The fields are explained in Table 1.

on X. Each host X runs a ROAM proxy PX that maintains a public trigger $(id, addr)$ where id is computed as a hash on X's home IP address, and $addr$ contains the current address of X and PX's port number, i.e., $[X.cip, PX.port]$. The proxy is responsible for updating the trigger every time the host's current IP address changes.

Figure 5 shows a typical data path in a legacy application, where a client CA running on host A is accessing a web server CB running on host B. (Figure 7 shows the pseudo-code executed by an ROAM proxy.) The source and the destination addresses in the headers of the packets sent by CA are the host IP addresses of A and B, respectively. Upon capturing the packet, PA encapsulates it inside $i3$ and proxy headers and sends it to CB through $i3$ using UDP. The identifier of the packet is set to B's public trigger identifier, i.e., $H(B.hip)$ (see function **ip_receive** in Figure 7). The format of the packets handled by the $i3$ proxies is shown in Figure 6.

When this packet arrives at B (see **i3_receive**), B's proxy (PB) strips off the $i3$ and proxy headers and forwards the packet to a local application. In addition, PB checks to see if the packet was sent to its own public trigger. If it was, then PB knows that A's proxy (PA) does not have a private trigger for B, so PB should send one. As an optimization, PB sets a timeout to see if it can piggyback the trigger on a packet sent from B's application (CB). Otherwise, when the timeout expires, B's proxy sends the private trigger in a separate packet. An end-host chooses private triggers on a per flow or a per communication peer basis. This precludes a malicious end-host from learning the private trigger used by (the flows of) another end-host and eavesdropping on it.

Assuming that CB does send a packet before the timeout expires, B's proxy piggybacks B's local private trigger on the outgoing packet to A. Since, B's proxy does not know A's private trigger, it uses A's public trigger (as $H(A.hip)$). When PA receives this packet, it inserts B's private trigger into its translation table with $B.hip$ as the key. In addition, PA sees that the packet was sent to its own public trigger, so it also sets a timeout and tries to piggyback its local private trigger to B.

When A changes its IP address from $A.cip$ to $A.cip'$ as a result of moving from one subnetwork to another, its ROAM proxy will remove the trigger containing the old IP address $A.cip$ and insert a trigger containing the new IP address $A.cip'$ into $i3$. The trigger identifier itself remains the same. Effectively, host mobility is masked by the $i3$ network from the communicating peer, and end-to-end connectivity is preserved.

While each end-host initially chooses its private triggers such that they are stored on nearby servers, end-hosts may eventually move far from those servers. To address this problem, each end-host can re-sample trigger servers either periodically or once it notices that its current private triggers are causing a high latency. The new private triggers can be ex-

```
// on receiving an IP packet p_ip from local applications
ip_receive(p_ip)
  p = i3_pkt_new();
  p.payload = p_ip;
  p.prox_hdr.flags = p.prox_hdr.flags ∨ DATA_MASK;
  // do we need to send a private trigger to the sender?
  if (exist_timeout(p_ip.dst_addr))
    p.prox_hdr.flags = p.prox_hdr.flags ∨ ID_MASK;
    p.prox_hdr.id = choose_private_trigger_id(p_ip.dst_addr);
    timeout_remove(p_ip.dst_addr);
  p.i3_hdr.id = i3_id(p_ip.dst_addr);
  i3_send(p);

// return the i3's identifier corresponding to addr
i3_id(addr)
  // get destination's private trigger from translation table,
  // if present
  if (exist_entry(trans_table, addr))
    return get_id(trans_table, addr);
  else
    return H(addr);
```

```
// on receiving an i3 packet p from network
i3_receive(p)
  p_ip = p.payload; // get encapsulated IP packet carried by p
  // does p carry sender's private trigger?
  if (p.prox_hdr.flags ∧ ID_MASK)
    update(trans_table, p_ip.src_addr, p.prox_hdr.id);
  else
    refresh(trans_table, p_ip.src_addr);
  // was p sent to the local host's public trigger?
  if (p.i3_hdr.id = H(p_ip.dst_addr))
    // p's source may not know our private trigger identifier ...
    timeout_set(p_ip.src_addr); // set a timeout to send it
  // does p contain data for a host's client?
  if (p.prox_hdr.flags ∧ DATA_MASK)
    ip_send(p_ip);

// timeout set by i3_receive for addr has expired
timeout(addr)
  p = i3_pkt_new();
  p.prox_hdr.flags = p.prox_hdr.flags ∨ ID_MASK;
  p.prox_hdr.id = choose_private_trigger_id(p_ip.dst_addr);
  p.i3_hdr.id = i3_id(p_ip.dst_addr);
  i3_send(p);
```

Figure 7: The pseudo-code executed by the proxy upon receiving packets from another host via $i3$ and from a host's client. The format of packet p handled by the proxy is given in Figure 6. *trans_table* denotes a translation table that stores the association between (1) a host IP address *addr*, and (2) the identifier of the private trigger inserted by the proxy running on host *addr*.

changed using a mechanism identical to the one used to exchange the original private triggers via the public triggers. The only change occurs in the **i3_receive** function: in addition to comparing the packet identifier to the the host's public trigger, we also compare it to the previous private trigger identifier, and then send out the new private trigger if necessary. This operation will be transparent to applications.

5 Implementation Details

The ROAM user-level proxy translates between existing Internet packets and $i3$ packets, and inserts/refreshes triggers on behalf of the applications. Applications do *not* need to be modified, and are unaware of the ROAM proxy. The ROAM proxy uses a virtual link-level interface (similar to [2]), called TUN[4], to transparently capture packets at user-level, and to hide host mobility from applications. The TUN virtual interface receives packets from user-level applications instead of from a physical media, and sends them to user-level applications instead of sending packets via physical media.

Users can specify a set of criteria, using the iptables tool, that determines whether a packet is redirected to the TUN virtual interface or passed directly to the IP routing table.

[4]The TUN virtual interface is implemented by the Universal TUN/TAP driver, which is included as a standard feature of the kernel in Linux 2.4 and later.

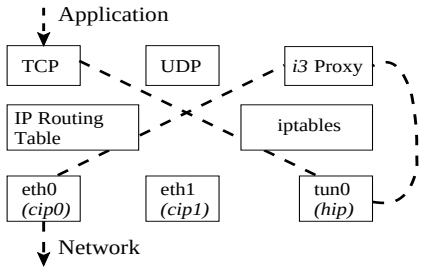


Figure 8: Data link, network, and transport layers on an end-host running the ROAM proxy software. The dashed line shows the path an outgoing TCP packet.

For example, if the user specifies the filter "-p udp –dport domain -j ACCEPT", then iptables will pass all DNS query and reply packets directly to the routing table.

Figure 8 illustrates the organization of our software when sending out a packet from the end host. The ROAM proxy reads and translates packets from tun0. To ensure that the translated packet does not get routed to tun0 again, the ROAM proxy adds a rule to iptables such that all packets from itself are passed directly to the routing table. Incoming packets from the correspondent host's proxy will arrive at the physical interface and be addressed to the ROAM proxy. The proxy will strip off the $i3$ and proxy headers and send it

to TUN, from which the applications will receive the packet (thus taking the reverse of the dashed path shown in Figure 8).

5.1 Multicast-based Soft Handoff

As a result of multicast-based soft handoff, the $i3$ server will send duplicate encapsulated packets to the MH. To prevent the MH's TCP/IP stack and applications from receiving duplicates of the inner packet, the ROAM proxy suppresses duplicates during multicast-based soft handoff.

The ROAM proxy maintains a small window of MD5 [23] digests of recent packets. The proxy computes digests over the first 20 bytes of the IP header and the first 8 bytes of the transport header. The first 28 bytes of a packet are sufficient to differentiate non-identical packets in practice [25]. To minimize duplicates, the window size must be sufficiently large so that a duplicated packet that arrives both via a very low latency link and via a very high latency link will be caught in the window. We use a window size of 1 second, which should be sufficient even if one path is very congested or contains a 500ms satellite link. Note that this scheme only eliminates duplicates generated by $i3$, and will *not* eliminate duplicate packets that are sent by the sender (e.g., TCP dupack). We show in Section 6 that a TCP bulk transfer flow using multicast-based soft-handoff achieves similar throughput to a flow without mobility.

Another implementation issue is when does the proxy stop using multicast. The algorithm we use is to remove an address when a large fraction of its packets are duplicates as this indicates that the address is redundant. The ROAM proxy maintains a counter of duplicate packets received on both addresses (d) and a counter of packets received on each addresses (p_i). When $d > \min(p_0/k, p_1/k)$, we simply remove the address that has received fewer packets in the last window. The value k is a constant indicating the fraction of an address's packets that must be duplicates before the address can be dropped. In addition, the proxy uses a timeout t to prevent a newly added address with poor connectivity from being removed until the timeout expires. In our implementation, we use $k = 2$ and $t = 30s$. We emphasize that this is a preliminary design; we are actively investigating alternate designs.

6 Evaluation

In this section, we present simulation and experimental results evaluating the benefit and cost of using ROAM.

6.1 Simulation

We use simulation to evaluate ROAM in comparison to Mobile IP with triangle routing, bidirectional routing, or route optimization. In doing so, we show that in all the mobility schemes, routing efficiency and fault tolerance are proportional to the amount of mobility infrastructure (either $i3$ servers in $i3$ or Home Agents in Mobile IP) deployed. However, for moderate amounts of infrastructure, ROAM provides much higher routing efficiency and fault tolerance than Mobile IP. In addition, ROAM's routing efficiency and fault tolerance scale with the amount of infrastructure devoted to it.

6.1.1 Methodology

We use our own `i3_mobility_sim` simulator to simulate $i3$ mobility and Mobile IP with its routing options. The purpose of the simulator is to measure routing latency while varying network topology, mobility routing schemes, mobility model, and communication model. The simulator simulates the creation, maintenance, and measurement of routes in the IP network, Mobile IP, and the $i3$ overlay network. It is not a packet level simulator.

Our simulation network topologies consist of three kinds of nodes: router nodes, mobility server nodes ($i3$ servers or Home Agents), and client nodes (mobile or correspondent hosts). We arrange the nodes according to the following topologies:

- A power-law random graph topology with 1779 router nodes, where the delay of each link is uniformly distributed in the interval $[5, 100)$ ms. Although power-law topologies have been shown [8] to accurately reflect Internet topology, we have not found a realistic model for assigning link latencies in a power law graph and so resort to a uniform latency distribution.

- A transit-stub topology generated with the GT-ITM topology generator [9] with 5000 nodes, where link latencies are 100 ms for intra-transit domain links, 10 ms for transit-stub links and 1 ms for intra-stub domain links. We assume that each stub node forms its own domain. Although transit-stub may not reflect Internet topology as accurately as power-law, we can assign link latencies in transit-stub in a way that reflects that intra-domain latencies are much lower than inter-domain latencies.

We define *domain* to be a group of nodes that have low latency links between them. For the power law topology, each node forms its own domain. For the transit-stub topology, each stub node forms its own domain. In both topologies, a varying number of mobility server nodes of up to 200% of the router nodes are randomly attached to the domains. There is little performance improvement for more than 200% server nodes because at that point, each domain is likely to have a server. A number of client nodes equal to the router nodes are also randomly attached to the router nodes. This gives a wide variety of possible client locations. For a particular topology, we run 50 different arrangements of client and server nodes to obtain sufficient variation in arrangement.

For a particular arrangement of nodes, we randomly select a home network (HN) from the client nodes with uniform distribution. Using that HN, we run 2000 arrangements of the mobile host (MH), and correspondent host (CH), as described below. This represents the number of locations a MH would visit before changing its HN, and is chosen to obtain sufficient variation in foreign locations.

We examine three mobility routing schemes in addition to regular IP routing: Mobile IP with triangular routing, Mobile IP with bidirectional tunneling, and ROAM mobility.

With the Mobile IP schemes, the MH also has a home agent (HA). The HA is usually assumed to be in the HN, but in a real network this may not be feasible because of deployment costs. In addition, requiring the HA to be in the HN restricts the number of MHs that can have mobility survive. Instead, we assume that a real network would have a more incremental deployment model, where a service provider would provide one or more home agents and map multiple users to each one. Therefore, in our simulations, we select the server closest to the HN as the HA.

With ROAM mobility, the MH uses the mobility-aware caching algorithm described in Section 4.1. The MH takes 32 samples in each move, maintains 10 entries in its cache, and replaces close entries when new samples are closer, but not less than 50% closer. These parameters are a compromise between performance and overhead because each sample consumes network bandwidth.

We simulate MH movement according to two mobility models: uniform and Pareto with respect to the HN. In the uniform model, the MH is uniformly randomly selected from the client nodes. In the Pareto home model, the probability that the MH is distance d from the HN is $1/d^2$. This simulates a MH that is close to the HN most of the time, but sometimes moves very far from the HN.

Similarly, we simulate communication with CHs according to three communication models: uniform, Pareto with respect to the HN, and Pareto with respect to the MH's current location in a foreign network. In the uniform model, the CH is randomly selected from the client nodes with uniform distribution. The Pareto home and Pareto foreign models assign distances to the CH according to the Pareto model given above, but relative to the HN or the MH's current location, respectively. These models simulate a CH that is close to the HN or MH, respectively, most of the time, but is sometimes very far from it.

Given all of these parameters, we measure the round trip time (RTT) of the various mobility schemes as shown in Figure 9. Note that in the ROAM case, both the MH and CH can be mobile, while in the triangular routing and bidirectional tunneling cases, we assume that the CH is stationary (i.e., the CH does not have a HA). If we were to assume that the CH is mobile, then the triangular routing and bidirectional tunneling cases would incur more latency, so this comparison favors those cases over ROAM.

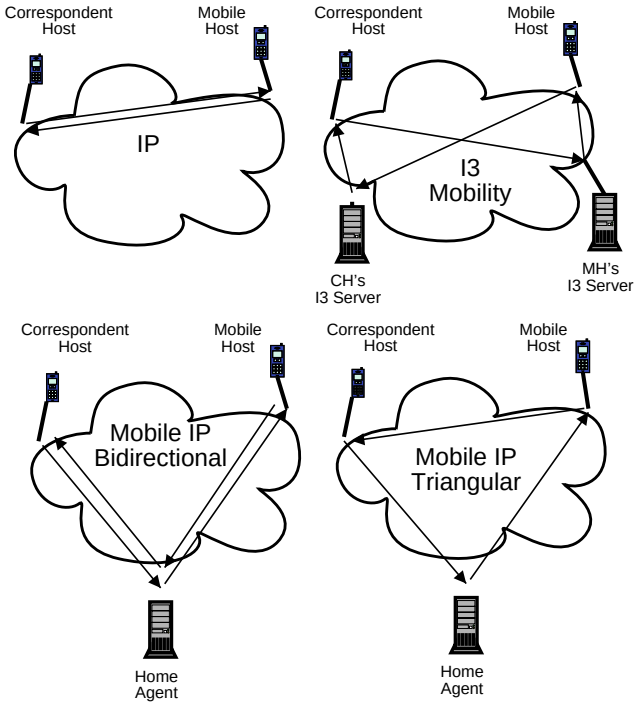


Figure 9: This figure shows the routes that packets travel in the different mobility schemes.

In all cases, we measure the latency stretch of the various schemes relative to the shortest IP path. Since there is considerable variance in the measurements due to the different random variables, we use the 90th percentile of the latency stretch to give a bound on the worst case performance.

6.1.2 Results: Stretch vs. Infrastructure

Figures 10 and 11 show a series of graphs which compare the 90th percentile stretch of MIP with triangular routing and bidirectional tunneling ("bi") and ROAM in a power law network (Figure 10) and a transit-stub network (Figure 11). Each graph shows a different combination of mobility model, communication model, and network topology.

In the power law network graphs, ROAM's stretch is always lower than that of MIP with bidirectional tunneling. This is because ROAM hosts are able to choose trigger servers closer to themselves than a home agent would be by using the mobility-aware caching algorithm (see Section 4.1). However, when there are few servers in the network, MIP triangular routing does better than ROAM. This is because ROAM hosts must use trigger servers to communication both from the CH to the MH and also in the reverse direction, while MIP with triangular routing only needs to use the home agent in the forward path. Small numbers of servers increase ROAM's stretch because it is less likely that a server will be

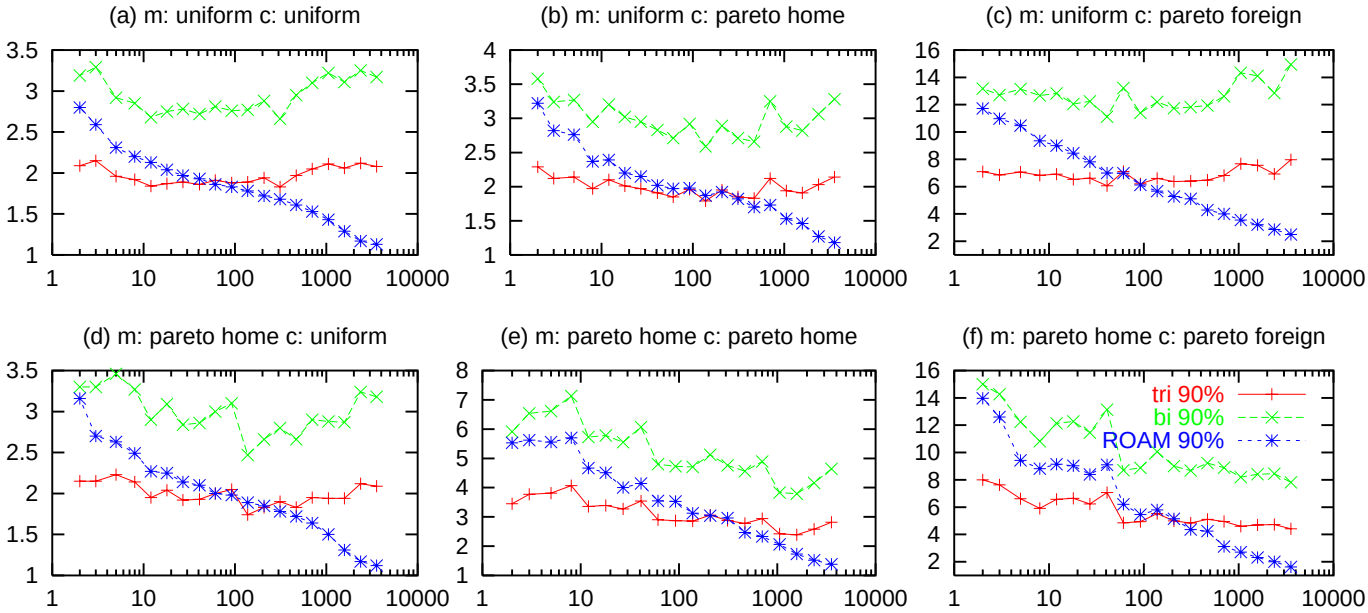


Figure 10: This figure compares the 90th percentile stretch of MIP with triangular routing ("tri") and bidirectional tunneling ("bi") with ROAM in a power law network. In each graph, the x-axis is the number of servers (home agents or $i3$ servers) on a log scale. The y-axis is the stretch relative to the shortest Internet path. Note that the different graphs have different scales on the y-axis. The different graphs are for varying mobility models ("m") and communication models ("c"). "uniform" indicates a random selection of a location in the network with uniform probability. "Pareto home" indicates a random selection of a location in the network with a Pareto distribution of distance from the home network. "Pareto foreign" indicates a random selection of a location in the network with a Pareto distribution of distance from the foreign network.

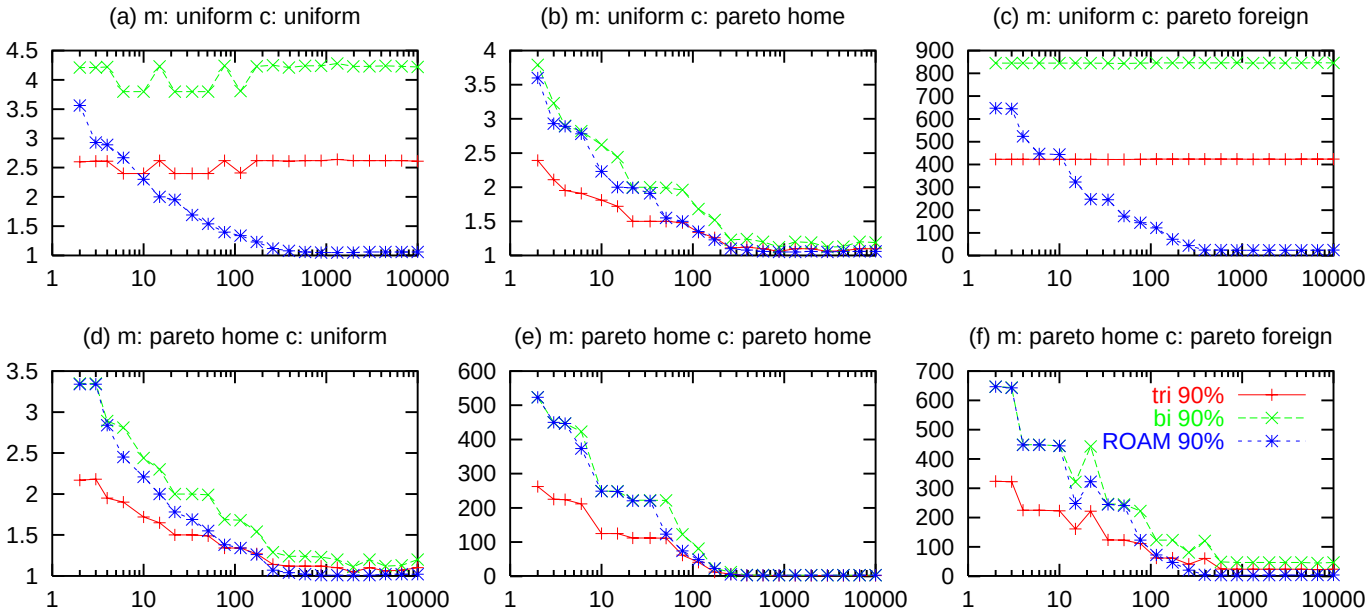


Figure 11: This figure is the same as Figure 10, except with a transit stub network.

nearby. For large numbers of servers, ROAM stretch is between 50% and 66% less than that of MIP with triangular routing. The crossover point varies from 100-500 servers depending on the mobility and communication model. 100-500 servers corresponds to 5-28% of the domains in the power law network having a local server.

When comparing MIP and ROAM in the transit-stub network, we find that when more than 1-2% of the transit-stub domains have a server, ROAM matches or exceeds MIP's stretch.

ROAM matches MIP when one or both of the communication end points (the CH and the MH) is close to the home network. We would expect that these are the optimal cases for MIP. Indeed, Figures 11 (b), (d), (e), and (f) show that MIP's stretch drops sharply as the number of deployed home agents increases. More home agents increase the likelihood that a HA will be in the HN, thus decreasing the stretch incurred by triangular routing or bidirectional tunneling when the CH and/or MH are close to the home network. However, the figures also show that ROAM's stretch converges with MIP's when more than 50 to 100 servers are deployed in the network (corresponding to 1-2% of the transit-stub domains having a server). This is because ROAM is able (through its trigger server caching algorithm) to dynamically find trigger servers which are as close to the MH and CH as a statically configured home agent.

ROAM significantly improves on MIP's stretch when neither the CH or the MH are close to the home network. We would expect that this is the worst case for MIP and Figures 11(a) and (c) validate this. Increasing the number of home agents in these cases does not decrease MIP's stretch because having a home agent close to the home network does not put it any closer to the CH or MH. In contrast, ROAM's stretch decreases as more servers are deployed because it can still dynamically find close trigger servers. Figure 11(a) shows that even when the CH, MH, and HN form a triangle with equal distribution of distance on each leg, ROAM's stretch is 40% that of MIP. When the CH and MH are Pareto close (as shown in Figure 11(c)), then ROAM has a stretch $1/400$th that of MIP with triangular routing. The difference is so large because the maximum latency in our transit-stub topology is over 1000ms while the minimum latency is only 1ms, so the impact of poor routing may be large.

The mobility and communication models make a greater difference in stretch in the transit-stub graphs than in the power law graphs. This is because in our transit-stub network locality affects latency more than in our power law network. Close nodes in a power law network have a higher latency between them than in a transit-stub network because the close transit-stub nodes are likely to be in the same stub domain. On the other hand, far nodes in a power network have a lower latency between them than in a transit-stub network because far transit-stub nodes are likely to be in different transit domains. Similarly, this is why MIP stretch decreases more in Figures 11 (b), (d), (e), and (f) than in Figures 10 (b), (d), (e), and (f).

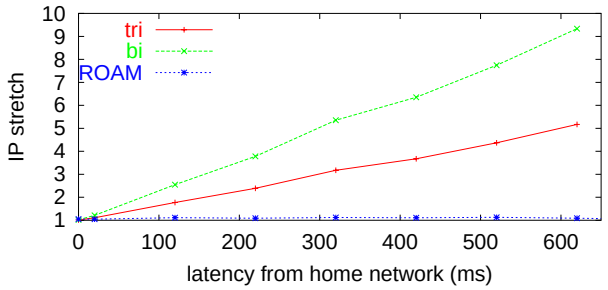


Figure 12: This figure compares the 90th percentile stretch of MIP with triangular routing ("tri") and bidirectional tunneling ("bi") with ROAM in a transit-stub network. The x-axis shows the latency from the home network in ms. The y-axis shows the stretch.

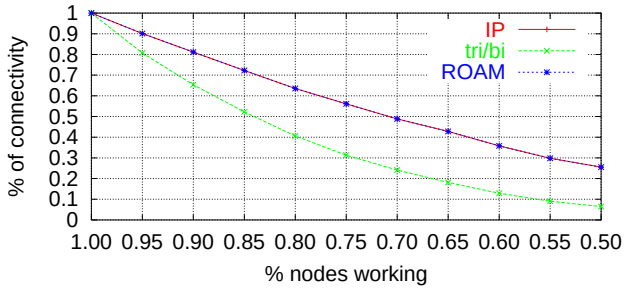


Figure 13: This figure compares the robustness of IP, MIP ("tri/bi"), and ROAM. The x-axis shows the probability that an end host, home agent, or $i3$ server is operational. The y-axis shows the probability that the CH and MH can communicate.

6.1.3 *Results: Stretch v.s. Distance from HN*

Instead of comparing the stretch to the number of mobility servers in the network, Figure 12 compares the stretch to the distance of the MH from the HN. This figure shows that as the distance from the home network increases, MIP's stretch increases linearly, while ROAM's stretch remains relatively constant. This simulation uses the transit-stub topology with 10000 servers, a uniform mobility model, and a uniform communication model.

6.1.4 *Results: Fault Tolerance*

In addition to stretch, we also simulate the failure of nodes. We vary the failure probability of the client and server nodes in the system from 0% to 50% and do 10,000 runs. In this simulation, we assume that both the MH and CH are mobile and have a home agent. We assume that IP routing succeeds when both the MH and CH are operational. We assume that MIP is functional when the MH, CH, MH's HA, and CH's home agent are operational. We assume that ROAM is func-

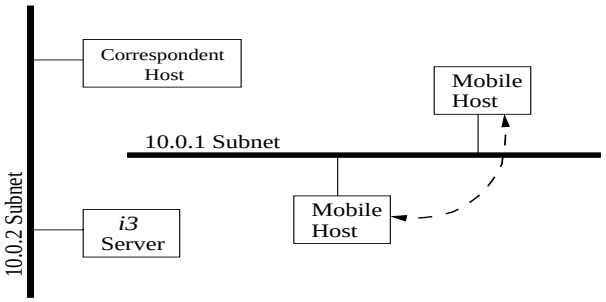


Figure 14: Network topology used for multicast-based soft handoff experiments. As shown by the dashed arrow, the mobile host moves between different locations on the 10.0.1 subnet during handoffs.

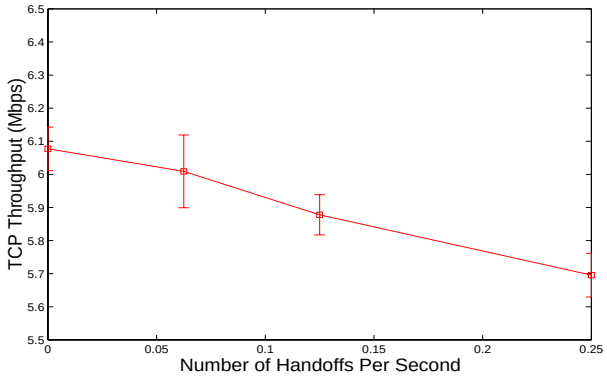


Figure 15: TCP throughput received by the MH as the handoff frequency increases. The vertical error bars show the standard deviations of the receiver throughput.

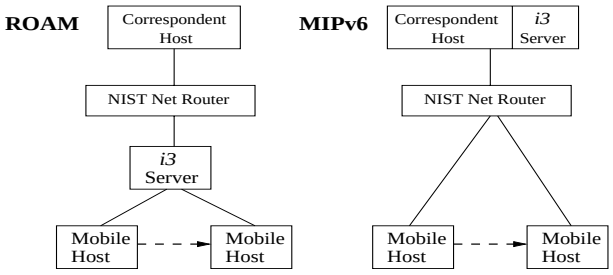


Figure 16: Network topology used for cold switch experiments. As shown by the dashed arrow, the mobile host moves from one location to another on the same subnet during handoff.

tional when the MH and CH are operational, and the MH and CH can both find an operational trigger server in their caches (of size 10).

Figure 13 shows the results of failing nodes on the likelihood of connectivity between the MH and CH. When nodes have a 5% chance of failing, MIP has an 85% likelihood of successful connectivity. When nodes have a 15% chance of failing, MIP likelihood of successful connectivity drops to only 50%. On the other hand, ROAM's robustness follows IP's regardless of the failure rate.

6.2 Experiments

In this section, we describe our test-bed and examine the effect of handoffs on TCP throughput.

6.2.1 Methodology

Our MH is a IBM Thinkpad T23 1.13GHz laptop running Red Hat Linux 7.3 with a 2.4.18 kernel. The CH is a 866 MHz desktop running a 2.4.18 Linux kernel. Our $i3$ server is a 800 MHz desktop running a 2.4.10 Linux kernel.

6.2.2 Results: Multicast-based Soft Handoffs

In this experiment, we perform TCP bulk transfers from the CH to the MH. Figure 14 shows our test-bed topology. The CH and the $i3$ server reside in the 100 Mb/s Ethernet 10.0.2 network[5]. The MH initiates TCP connections from one location on the 10.0.1 subnet, and then moves to another location on the same subnet at a later point, or vice versa. Both MH locations use identical connections with 10Mbps links. The purpose of this simple configuration is to expose the performance impact of multicast-based soft handoffs. Each run involved a 16 seconds TCP bulk transfer and we varied the number of handoffs (0, 1, 2, and 4) performed during a transfer. This was repeated ten times at each handoff frequency. Figure 15 plots TCP throughput received by the MH as the number of handoffs increases during the bulk transfer. The vertical error bars show the standard deviation of the receiver throughput.

We see that as handoff frequency increases, the TCP throughput degradation is minimal. In fact, there are no losses across the multicast-based soft handoffs as both interfaces are available. The slight performance penalty is caused by the overhead of MD5 digest computation of every packet received and detection of duplicates during handoffs. This demonstrates the effectiveness of ROAM to support rapid handoffs. For example, consider a user on an airplane moving at 500 miles per hour, and cell coverage sizes with diameters of 1.5 miles. In this case, the user makes 5 cell crossings per minute, which can be easily supported by ROAM. To support multiple such users on the airplane, we can use a NAT-like device to aggregate cell-crossings made by users, and thereby alleviate the handoff load on the $i3$ trigger server.

6.2.3 Results: Cold Switch

In this experiment, we compare the handoff performance of ROAM and MIPv6 during a cold switch when the MH is far away from the CH. Figure 16 shows the experimental setup. We use the NIST Net [19] network emulation package to emulate a round trip time (RTT) of 70 milliseconds between the MH and the CH. In the setup for ROAM, RTT between the

[5]Network addresses are anonymized.

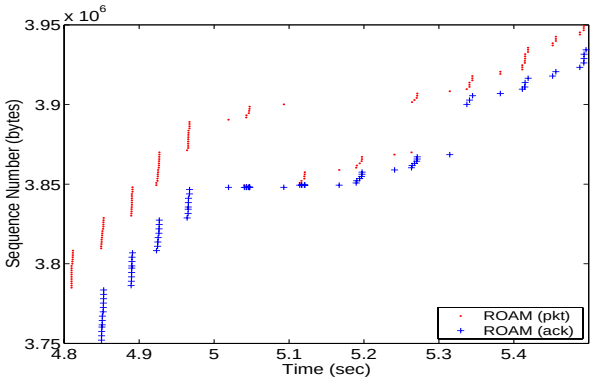


Figure 17: TCP sequence trace showing a bulk transfer with a cold-switch for ROAM.

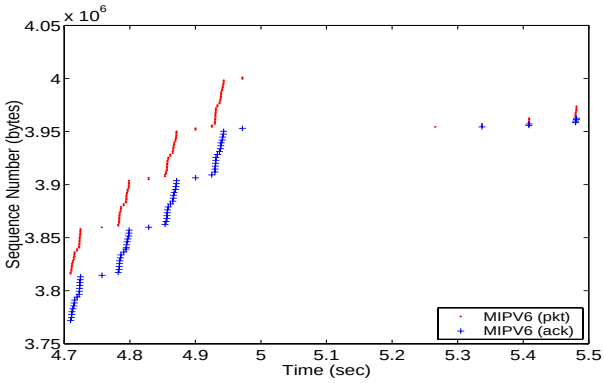


Figure 18: TCP sequence trace showing a bulk transfer with a cold-switch for MIPv6.

MH and the i3 server is approximately 3 milliseconds. The nistnet router delays packets between the i3 server and the CH by 70 milliseconds. We emulate the MIPv6 scenario by running the i3 server on the same machine as the CH since binding updates are propagated to the CH in MIPv6. The nistnet router delays packets between the MH and the CH by 70 milliseconds. During a cold switch, the first interface is shutdown around 35-40 milliseconds before the second interface is brought up. During this disconnected interval, packets from the $i3$ server to the MH are lost in both ROAM and MIPv6. However, the number of packets that are lost after cold switch completes is proportional to the delay between the MH and the indirection point.

Figures 17 and 18 plot the TCP sequence numbers seen at the CH (TCP sender) for the ROAM and MIPv6 scenarios during a cold switch. ROAM recovers from packet loss caused by the cold switch by entering fast retransmit when the MH receives duplicate acknowledgements generated by packets received after the lost packets. However, in MIPv6, the MH loses the entire window of data and the CH waits for a timeout and goes into slow start before retransmitting the lost packets.

Routing	Header Size	Relative Overhead	Transmission Delay
IP	40B	1.25	23ms
Mobile IP	60B	1.88	27ms
ROAM	117B	3.66	41ms
ROAM w/comp	45.8B	1.43	24ms

Table 2: This table shows the header overhead of various routing schemes. All header sizes are in bytes. The listed overhead is relative to a 32 byte payload. The transmission delay is for the given header size, a 32 byte payload, and a 32Kb/s link bandwidth.

If the disconnectivity time due to cold switch is t, and $t < RTT < 2t$, then ROAM can recover by fast retransmit whereas MIPv6 has to recover by timeout. If RTT is greater than $2t$, then both ROAM and MIPv6 can recover through fast retransmit. However, ROAM will recover sooner because of its ability to choose a nearby i3 server irrespective of the CH's location, thereby greatly reducing packet loss.

7 Discussion

In this section, we discuss some important security issues and the overhead of ROAM. We then discuss the possibility of using ROAM to exchange only control information, while data packets are forwarded via IP. Finally, we discuss the possibility to replace the ROAM proxy with a NAT-like solution, and some deployment issues.

Trigger hijacking. As discussed in Section 4.4, a public trigger's identifier can have only one address associated with it. Although this ensures that no one can eavesdrop on the communication destined to a public trigger, an attacker can wait for a host to fail to refresh its public trigger, and insert its own trigger with the same identifier. As a result, all packets destined to that identifier will be received by the attacker. This attack is similar to hijacking a DNS entry. One solution is to eliminate the EXCLUSIVE_ID bit and then use public key cryptography to protect against eavesdropping. When initiating a connection, A encrypts its private trigger id_a under the public key of B, before sending it to B via B's public trigger id_p. Since A's private trigger is encrypted, a malicious user M cannot impersonate B even if it inserts a trigger $(id_p, addr_m)$ into $i3$.

The public key of B can be obtained through several possibilities. To name a few, one can use a Public Key Infrastructure, store public keys in the DNS system, or obtain it out-of-band similar to what ssh does.

Overhead. Although the $i3$ encapsulation used by ROAM adds packet header overhead, ROAM with header compression adds only 18% more overhead than standard TCP/IP for a 32 byte payload. This would add only 4% more delay on a 5ms latency, 32Kb/s bandwidth link.

Table 2 lists the overhead of various routing schemes relative

to different payload sizes. A standard web browser using IP and TCP or an IP telephony application using IP, UDP, and RTP has a total header size of 40 bytes. Mobile IP requires 20 additional bytes for IP in IP encapsulation. The size of the $i3$ header in the current implementation is 48 bytes (of which 32 bytes is the $i3$ identifier). The proxy header has a minimum size of one byte (see Figure 6). The encapsulating IP and UDP headers total 28 bytes. Thus, the ROAM total header size is 28 (encapsulating packet) + 1 (proxy) + 48 ($i3$ header) + 40 (original packet) = 117. When private identifiers are piggybacked in the data packets (typically only in the beginning of the connection), the overhead increases by another 32 bytes.

However, header compression has been shown [14] to reduce packet header overhead by a factor of 5. If we compress the 89 bytes of header after the encapsulating header (which must remain uncompressed to route through the Internet), then we reduce the total header size to an average of 45.8 bytes. This only requires modifications to the proxy and $i3$ server software.

Table 2 shows that even for a 32 byte IP telephony payload, the ROAM compressed header overhead is only 18% greater than that of standard TCP/IP. On a hypothetical 5ms latency, 32Kb/s link, the net difference in transmission delay is 5%. This overhead decreases as packet sizes, latencies, and bandwidths increase.

Another source of overhead is the user level proxy which causes each packet to traverse the OS–user level boundary twice. This can reduce the maximum throughput that can be achieved by the end host. However, that maximum is unlikely to be reached even in a relatively high bandwidth wireless network like 802.11b (11Mb/s). If this becomes an issue, the proxy can be eliminated at the cost of implementing its functionality in the kernel.

Control plane indirection. We assume that all packets are transmitted via $i3$. For most applications we expect the indirection overhead to be acceptable, but there might be applications for which achieving the highest possible throughput and lowest latency is critical. For those applications, one can implement a solution similar to TCP Migrate, where $i3$ is used only to exchange the new IP addresses when end-hosts move. In comparison to the basic TCP Migrate solution, such an approach would allow simultaneous mobility and would avoid overloading the DNS.

Home proxy. We assume that each end-host runs a ROAM proxy. In some cases, the robustness and efficiency this provides may not be worth the management and deployment costs. For example, during initial deployment, few of a mobile host's correspondent hosts will have ROAM proxies. An alternative is to deploy a home proxy for a mobile host that implements the functionality of the ROAM proxy for all of its non-ROAM correspondent hosts. This home proxy is analogous to the Home Agent in Mobile IP in that it is only used for hosts that cannot use a more efficient routing method.

Deployment issues. Our initial design assumes that ROAM uses a shared overlay infrastructure ($i3$). The most likely deployment strategy of such an infrastructure is still unclear. Options include a single provider for-profit service (like content distribution networks), a multi-provider for-profit service (like ISPs), and a cooperatively managed nonprofit infrastructure (like Gnutella). While full deployment is always hard to achieve, our solution is incrementally deployable; if the efficiency and robustness are not a concern, then it could start as a single server. Moreover, it does not require the cooperation of ISPs, so third parties can provide this service.

Personal/Session mobility. Our focus on using ROAM for network layer mobility allows only a brief description of how the same architecture can be used for personal/session mobility (Section 2). Personal/session moblity requires tracking the set of active devices for a user and routing to the optimal device. Devices register themselves for a particular person whenever they detect an authorized user is nearby (e.g., devices have Bluetooth transceivers and users carry Bluetooth smart cards). Devices register by setting a trigger with an identifier representing that user. Given multiple simultaneously registered devices, the devices follow an agreement protocol to decide which one will handle a particular session (e.g., the least expensive one or the highest performance one). Leveraging the ROAM infrastructure for personal/session mobility removes the cost of deploying infrastructure specific to any particular application or personal/session mobility scheme.

Service composition. With $i3$, endhosts can easily redirect data packets through transcoding servers before reaching the destination. For example, a low-bandwidth wireless mobile host can route a MPEG stream to go through a MPEG-to-H.263 transcoder, and receive a H.263 stream instead. Furthermore, a mobile host can redirect its packets to go through performance optimization agents such as TCP snoop [3] to improve performance over wireless links. Refer to [27] for details on how $i3$ uses a stack of identifiers to achieve service composition.

8 Conclusion

In this paper we present a highly robust and efficient mobility architecture. ROAM uses an indirection infrastructure which is implemented as an overlay network. This infrastructure allows end hosts to avoid the inefficiency of triangle routing by allowing them to choose nearby indirection points. This infrastructure is as robust as the underlying IP network. ROAM preserves location privacy and allows simultaneous mobility while not requiring any changes to the TCP/IP protocol stack.

Simulation results show that our solution has a low latency stretch and it is highly robust compared to Mobile IP. We evaluate a prototype of ROAM in a small testbed. Preliminary experimental results using this prototype demonstrate that ROAM provides good support for soft-handoff and frequent mobility.

However, more remains to be done to evaluate our solution better. To aid with this, we plan to deploy ROAM on a larger scale with end hosts and $i3$ servers spanning the continental US.

References

[1] ANEROUSIS, N., GOPALAKRISHNAN, R., KALMANEK, C., KAPLAN, A., MARSHALL, W., MISHRA, P., ONUFRYK, P., RAMAKRISHNAN, K., AND SREENAN, C. Tops: An architecture for telephony over packet networks. *IEEE Journal on Selected Areas in Communications 17*, 1 (1999), 91–108.

[2] BAKER, M. G., ZHAO, X., CHESHIRE, S., AND STONE, J. Supporting Mobility in MosquitoNet. In *Proceedings of the 1996 USENIX Technical Conference* (Jan. 1996).

[3] BALAKRISHNAN, H., SESHAN, S., AMIR, E., AND KATZ, R. H. Improving tcp/ip performance over wireless networks. In *Proceedings of MOBICOM (nov 1995)*.

[4] CALHOUN, P., JOHANSSON, T., AND PERKINS, C. Internet Draft: Diameter Mobile IPv4 Application. Internet Engineering Task Force, june 2002. work in progress.

[5] CHESHIRE, S., AND BAKER, M. Internet Mobility 4x4. In *Proceedings of SIGCOMM* (1996), pp. 318–329.

[6] DABEK, F., KAASHOEK, F., KARGER, D., MORRIS, R., AND STOICA, I. Wide-area cooperative storage with cfs. In *Proc. ACM SOSP'01* (Banff, Canada, 2001), pp. 202–215.

[7] DEERING, S., AND CHERITON, D. R. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems* (May 1990).

[8] FALOUTSOS, M., FALOUTSOS, P., AND FALOUTSOS, C. On Power-law Relationships of the Internet Topology. In *Proceedings of SIGCOMM* (1999), pp. 251–262.

[9] Georgia tech internet topology model. `http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html`.

[10] GUSTAFSSON, E., JONSSON, A., AND PERKINS, C. E. Internet Draft: Mobile IP Regionalized Tunnel Management. Internet Engineering Task Force, august 1999. work in progress.

[11] HELMY, A. A Multicast-based Protocol for IP Mobility Support. In *ACM SIGCOMM Second International Workshop on Networked Group Communication* (2000), pp. 49–58.

[12] HELMY, A., AND JASEEMUDDIN, M. Efficient Micro-Mobility using Intra-domain Multicast-based Mechanisms (M&M). Tech. rep., USC, aug 2001.

[13] JOHNSON, D. B., AND PERKINS, C. Internet Draft: Mobility Support in IPv6. Internet Engineering Task Force, june 2002. work in progress.

[14] LILLEY, J., YANG, J., BALAKRISHNAN, H., AND SESHAN, S. A unified header compression framework for low-bandwidth links. In *Proceedings of MobiCOM* (2000), pp. 131–142.

[15] MALKI, K., CALHOUN, P., HILLER, T., KEMPF, J., MCCANN, P., SINGH, A., SOLIMAN, H., AND THALANANY, S. Internet Draft: Low Latency Handoffs in Mobile IPv4. Internet Engineering Task Force, june 2002. work in progress.

[16] MANIATIS, P., ROUSSOPOULOS, M., SWIERK, E., LAI, K., APPENZELLER, G., ZHAO, X., AND BAKER, M. The Mobile People Architecture. In *ACM Mobile Computing and Communications Review* (July 1999).

[17] MYSORE, J., AND BHARGHAVAN, V. A New Multicasting-Based Architecture for Internet Host Mobility. In *Proceedings of ACM/IEEE MobiCom* (1997), pp. 161–172.

[18] MYSORE, J., AND VADUVUR, B. Performance of transport protocols over a multicasting-based architecture for internet host mobility. In *IEEE International Conference on Communications* (1998), pp. 1817–1823.

[19] Nist net network emulator. `http://snad.ncsl.nist.gov/itg/nistnet/index.html`.

[20] PERKINS, C. RFC 2002: IP Mobility Support. Internet Engineering Task Force, oct 1996.

[21] PERKINS, C., AND JOHNSON, D. B. Internet Draft: Route Optimization in Mobile IP. Internet Engineering Task Force, sept 2001. work in progress.

[22] RITTER, M., FRIDAY, R. J., GARCES, R., FILIPPO, W. S., AND NGUYEN, C.-T. Mobile Connectivity Protocols and Throughput Measurements in the Ricochet MicroCellular Data Network (MCDN) System. In *Proceedings of MobiCom* (aug 2001).

[23] RIVEST, R. RFC 1321: The MD5 message-digest algorithm. Internet Engineering Task Force, apr 1992.

[24] SNOEREN, A. C., AND BALAKRISHNAN, H. An End-to-End Approach to Host Mobility. In *Proceedings of MobiCom* (aug 2000).

[25] SNOEREN, A. C., PARTRIDGE, C., SANCHEZ, L. A., JONES, C. E., TCHAKOUNTIO, F., KENT, S. T., AND STRAYER, W. T. Hash-Based IP Traceback. In *Proceedings of SIGCOMM* (aug 2001).

[26] SOLIMAN, H., CASTELLUCCIA, C., EL-MALKI, K., AND BELLIER, L. Internet Draft: Hierarchical MIPv6 mobility management. Internet Engineering Task Force, july 2002. work in progress.

[27] STOICA, I., ADKINS, D., ZHUANG, S., SHENKER, S., AND SURANA, S. Internet indirection infrastructure. In *Proceedings of SIGCOMM (aug 2002)*.

[28] TANG, D., AND BAKER, M. Analysis of a metropolitan-area wireless network. In *Proceedings of MobiCom* (1999), pp. 13–23.

[29] VAKIL, F., DUTTA, A., CHEN, J.-C., TAUIL, M., BABA, S., NAKAJIMA, N., SHOBATAKE, Y., AND SCHULZRINNE, H. Internet Draft: Supporting Mobility for TCP with SIP. Internet Engineering Task Force, december 2000. work in progress.

[30] WANG, H. J., RAMAN, B., NEE CHUAH, C., BISWAS, R., GUMMADI, R., HOHLT, B., HONG, X., KICIMAN, E., MAO, Z., SHIH, J. S., SUBRAMANIAN, L., ZHAO, B. Y., JOSEPH,

A. D., AND KATZ, R. H. Iceberg: An internet-core network architecture for integrated communications. *IEEE Personal Communications Magazine* (2000).

[31] YEGIN, A., PERKINS, C., TSIRTSIS, G., DOMMETY, G., MALKI, K., AND KHALIL, M. Internet Draft: Fast Handovers for Mobile IPv6. Internet Engineering Task Force, march 2002. work in progress.