Host Mobility Using an Internet Indirection Infrastructure

Shelley Zhuang Kevin Lai Ion Stoica Randy Katz Scott Shenker

Abstract

We propose the Robust Overlay Architecture for Mobility (ROAM) to provide seamless mobility for Internet hosts. ROAM is built on top of the Internet Indirection Infrastructure ($i3$). With $i3$, instead of explicitly sending a packet to a destination, each packet is associated with an identifier. This identifier defines an *indirection* point in $i3$, and is used by the receiver to obtain the packet.

ROAM takes advantage of end-host ability to *control* the placement of indirection points in $i3$ to provide efficient routing, fast handoff, and preserve location privacy for mobile hosts. In addition, ROAM allows end hosts to move simultaneously, and is as robust as the underlying IP network to node failure. We have developed a user-level prototype system on Linux that provides transparent mobility without modifying applications or the TCP/IP protocol stack. Simulation results show that ROAM's latency can be as low as 0.25-40% of Mobile IP. Experimental results show that with soft handoff the TCP throughput decreases only by 6% when there are as many as 0.25 handoffs per second.

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) will soon provide the potential for ubiquitous, though heterogeneous, coverage. To realize this potential, users 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). These services would enable users to run applications such as IP telephony, instant messaging, and audio streaming while mobile.

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 [1]), redirecting through the DNS (e.g., TCP Migrate [2]), or using indirection at the link layer (e.g., cellular mobility schemes).

However, these proposals lack one or more of the following properties to fully realize the promise of ubiquitous mobility:

- *Efficient routing*: packets should be routed on paths with latency close to the shortest path provided by IP routing.
- *Efficient handoff*: the loss of packets during handoff should be minimized and avoided, if possible.
- *Fault tolerance*: communication between mobile hosts should not be more vulnerable to faults than communication between stationary hosts.
- *Location privacy*: the host's topological location should not be revealed to other end-hosts.
- *Simultaneous mobility*: end hosts should be able to move simultaneously without breaking an ongoing session between them.
- *Personal/session mobility*: a user should be able to redirect a new session or migrate an active one from one application or device to another one when a better choice becomes available [3, 4, 5].
- *Link layer independence*: users should be able to seamlessly operate across heterogeneous link layer technologies, not all of which support the same link layer mobility scheme (e.g., GSM mobility).

In this paper, we propose (to the best of our knowledge) the first solution to achieve all of these properties. Our solution, called Robust Overlay Architecture for Mobility (ROAM), is built on top of the Internet Indirection Infrastructure ($i3$) [6]. $i3$ is implemented as an overlay network on top of IP, 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 an $i3$ node (server). 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. For instance, 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. Section 4 discusses in detail how ROAM achieves the above properties.

At the architectural level, this paper makes two contributions. First, it demonstrates the benefit of giving end-hosts *control* on the placement of the indirection points. This allows, end-hosts to optimize the routing and handoff efficiency. Second, it demonstrates the benefits of a mobility architecture based on a shared overlay network. Such a solution leverages the robustness of the overlay networks.

In addition, 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. Section 2 presents the related work, and Section 3 gives an overview of $i3$. Section 4 discusses the design of ROAM, and Section 5 presents the ROAM support for legacy applications. Section 6 presents some implementation details. Section 7 presents simulation and experimental results. Finally, Section 8 discusses some open issues, and Section 9 concludes the paper.

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 [7], 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 [1] and IPv6 [8] 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 [9] 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 [10], 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 *latency 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.

To address routing anomalies and robustness issues associated with a fixed HA, researchers have proposed the notion of dynamic home agents in MIPv4 [11]. 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 [8] 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.

Recently, two mechanisms have been proposed to increase handoff performance in MIPv4 and MIPv6: (1) low latency handoff [12], and (2) fast handover [13]. 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 [14] and Hierarchical Mobility [15] 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 homogeneous link-layer technology. In contrast to both these mechanisms, ROAM supports fast handoff by giving end-hosts implicit control over trigger placement.

The Host Identity Protocol [16] supports mobility by decoupling the transport from the network layer, and binding the transport to a host identity. Similarly, Location Independent Networking for IPv6 [17] specifies a unique identifier for a host regardless of its location. These ideas are in line with the seminal work by Jerome Saltzer on host identifier and locator separation [18]. However, $i3$ provides a general-purpose indirection infrastructure that enables a variety of communication services beyond mobility, such as multicast, and transcoding. Section 4.2.2 will present the use of $i3$ multicast to support soft handoffs for mobile hosts.

Supporting Mobility for TCP with SIP [19] spoofs constant TCP endpoints in a similar way to MIP with route optimization. This requires modifying the IP stack of the CH.

The Mobility Support using Multicasting in IP (MSM-IP) architecture [20, 21] implements mobility using IP Multicast [22]. The main advantages of MSM-IP are that it can have low latency and do handoffs with little or no packet loss. Several studies [21] [23] [24] have shown that multicast mo-

bility can cut the latency 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 [2], 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, which 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. However, 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) [4] ICEBERG [5], and Telephony Over Packet networkS [3]) 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 or not. 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).

In contrast to all of the above schemes, the novelty of our approach is the use of an overlay indirection infrastructure that gives endhosts control over the placement of the indirection points. As a result, ROAM achieves 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 as well as sessions and people.

3 Background

In this section we present a brief overview of an Internet Indirection Infrastructure, $i3$ [6], 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 a rendezvous-based communication abstraction. In its simplest form, a packet is a pair $(id, data)$

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

(a)

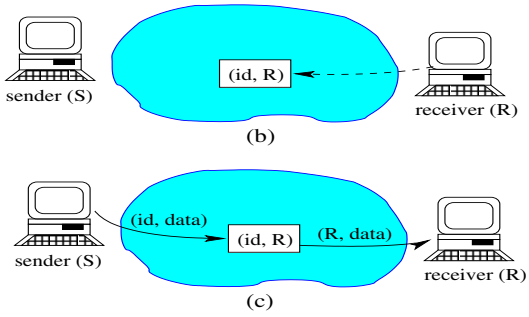


Figure 1: (a) $i3$'s API. Example illustrating communication between two nodes: (b) Receiver R inserts trigger (id, R). (c) Sender S sends packet $(id, data)$.

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 its simplest form, a trigger is a pair $(id, addr)$, 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. R 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.

Thus, as in IP multicast, 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 [6].

3.2 $i3$ Implementation

$i3$ is implemented as an overlay network composed of

[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.

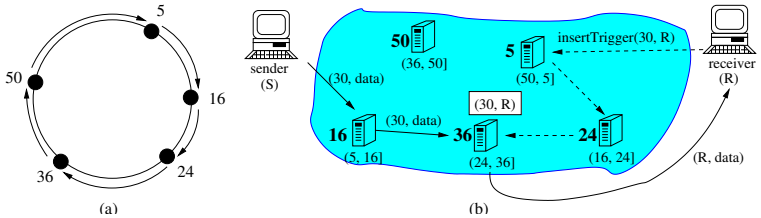


Figure 2: (a) A Chord identifier circle for $m = 6$, with 5 servers identified by 5, 16, 24, 36, and 50, respectively. (b) Receiver R inserting trigger $(30, R)$, and sender S sending packet $(30, data)$.

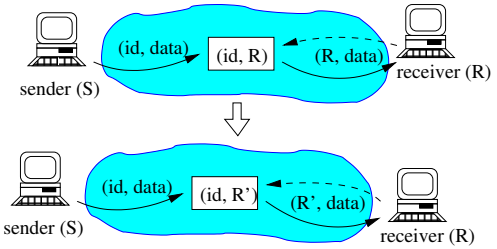


Figure 3: Upon changing its address from R to R', a receiver needs only to update its trigger. This change is transparent to the sender.

servers that store triggers and forward packets.

To maintain this overlay network and to route packets in $i3$, we use the Chord lookup protocol [25]. 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 at 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 an 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 [25]. 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 must periodically refresh 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 on top of $i3$.

Achieving host mobility using $i3$ 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:

Efficient routing. ROAM takes into account the mobility pattern of the end-hosts to improve $i3$'s ability to provide low latency stretch (see Section 4.1).

Efficient handoff. ROAM implements fast handoff and multicast-based handoff to reduce packet loss during handoffs (see Section 4.2).

Fault tolerance. 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.[2] Section 7.1.4 evaluates ROAM's robustness through simulation.

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

Location privacy. ROAM allows end-hosts the flexibility of choosing triggers to not reveal any location information. Section 4.3 discusses the tradeoffs between location privacy and routing efficiency.

Personal/session mobility. Unlike Mobile IP, ROAM 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 (see Section 4.4).

Next, we describe efficient routing, efficient handoff, location privacy, and personal/session mobility in more detail. The key to achieving these properties is the ability of end-hosts to *control* the location of a trigger in $i3$. Since fault tolerance and simultaneous mobility follow directly from $i3$'s properties, we do not discuss them here.

[2] To make $i3$ server failure completely transparent to end-hosts, $i3$ can replicate triggers [6].

4.1 Efficient Routing

Although the Chord lookup protocol limits the number of hops traversed in $i3$ 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. To deal with this problem, $i3$ uses two techniques: (1) trigger server caching, and (2) trigger sampling [6]. With the first technique, an end-host caches the server storing a particular trigger, and then sends trigger refresh messages and data packets matching the trigger to that server directly via IP. For example, in Figure 2, both the sender (S) and the receiver (R) would cache server[3] 36.

While caching ensures that subsequent packets will traverse only one $i3$ server, the delay can still be large if that server is far from both end-hosts. To address this problem, end-hosts use trigger sampling to pick triggers stored at nearby servers. An end-host picks triggers with random identifiers, measures the round trip delay to the servers that store those triggers, and then uses the trigger with the lowest delay. Note that an end-host only needs to sample at most once per location change, and not every time it opens a connection. As shown in [6], this method is quite efficient. In an $i3$ system with 2^{16} servers, taking only 32 samples results in a 90th percentile latency stretch of only 1.5.

Next, we present an extension of these techniques that takes into account the movement pattern of mobile hosts.

4.1.1 Mobility-Aware Trigger Caching

We assume that mobile hosts are likely to move in a pattern where some moves are short (in geographic distance and network latency), but some moves are very far [26]. 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. This pattern also fits a user who moves among different network technologies with widely varying network latency.

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.

In Section 7.1, we show by simulation that this caching scheme can reduce the latency stretch to nearly 1.0.

[3]Since the trigger can be reused across connections, the $O(\log n)$ traversal only needs to be done when $i3$ servers fail or when using a trigger for the first time.

4.2 Efficient Handoff

To reduce the loss of packets during handoffs, ROAM supports fast handoff and multicast-based handoff.

4.2.1 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 latency from the MH to the HA or CH is high. If the MH stops receiving packets at the old IP address before starting to receive packets at the new address (*cold switch*), then those in-flight packets will be lost.

With ROAM, end-hosts can alleviate this problem by choosing 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, this choice will reduce packet loss. In Section 7.2.2, we use experiments to compare the performances of ROAM and MIPv6 with cold-switching. See section 2 for a qualitative comparison of our approach to two recently proposed mechanisms to increase the performance of handoff in MIPv4 and MIPv6 [12, 13].

4.2.2 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

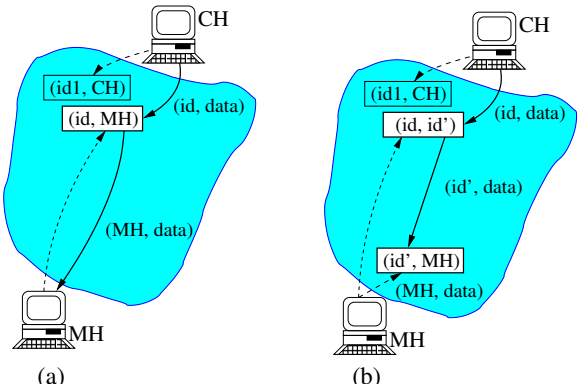


Figure 4: Achieving location privacy: (a) MH chooses a trigger close to the CH; (b) MH uses two triggers to preserve fast handoff.

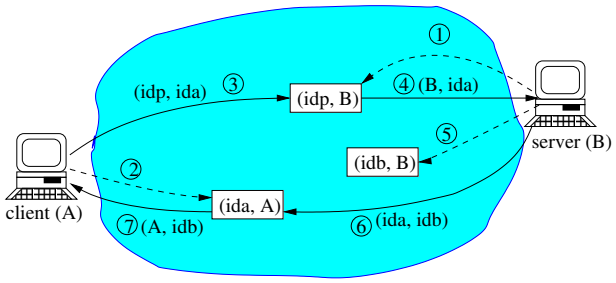


Figure 5: Example of setting up a connection via private triggers id_a and id_b between two hosts A and B. id_p is B's public trigger.

the MH to take advantage of the best available connectivity. There are two things worth noting. First, the use of multicast is completely transparent to the sender. Second, fast handoff is still necessary for cases when the MH cannot listen simultaneously at both addresses. For example, an 802.11b client cannot be simultaneously connected to two base stations on different channels [27]. We address the problems of determining when to stop using multicast and how to suppress duplicate packets in Section 6.1. We discuss the implication of multicast on communication privacy in Section 8.

4.3 Location Privacy

While choosing a trigger (id, MH) at an $i3$ server close to the MH improves the routing and handoff efficiency, this choice would reveal (to some extent) the location of the MH. To avoid this problem, the MH can choose id such that the trigger is stored at an $i3$ server close to the CH instead of itself. This would result in a low latency stretch without compromising the MH's location privacy. Let (id_1, CH) be the trigger advertised by the CH to the MH (see Figure 4(a)). Assuming that the CH chooses this trigger close to itself, the MH can simply choose id to share the first 128 bits with id_1. This is because with $i3$, all triggers whose identifiers share the same 128-bit prefix are stored at the same $i3$ server [6].

To also allow fast handoff, the MH can use two triggers, one of the form (id, id') [4] inserted near the CH, and one of the form (id', MH) inserted close to itself (see Figure 4(a)). Note that this change is transparent to the CH, i.e., the CH will still send packets of the form $(id, data)$ to the MH. Because the CH does not need to know id', the location privacy of the MH is ensured. Moreover, the choice of id and id' ensures a low latency stretch, and enables the MH to do fast handoff by updating trigger (id', MH).

If both end-points require location privacy, they can choose completely random $i3$ servers. The flexibility of $i3$ allows each application to make the tradeoff between location privacy and routing efficiency as desired.

[4]Note this is a generalized form of triggers, which allows a trigger to send a packet to another identifier rather than to an address.

4.4 Personal/Session mobility

While the focus of this paper is on using ROAM for network layer mobility, ROAM also provides support for personal/session mobility (Section 2). Personal/session mobility 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) [28]. 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 performing 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.

5 Application Support

One of the central goals of ROAM is to support legacy applications. Ideally, this would allow us to transparently run existing applications on top of ROAM. In this section, we first describe how $i3$ can support native applications, and then present our solution for legacy applications.

5.1 Support for Native Applications

So far we have assumed that each end-host is free to choose its triggers independently. The natural question is how does an end-host learn about the trigger of another end-host? To answer this question, $i3$ introduces the concept of *public* and *private* triggers [6]. 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 public trigger of the "New York Times" web server can be the hash of its name.

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 7)
$i3_hdr.id$	$i3$ 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 5.2.

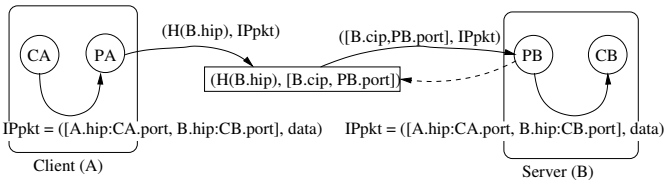


Figure 6: Legacy application support. $A.hip$ and $B.hip$ represents home IP addresses of hosts A, and B, respectively. Each host has a proxy that intercepts applications packets and send them via $i3$.

Consider the application in Figure 5 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. The important point to note here is that end-hosts have full control on selecting their private triggers.

5.2 Support for Legacy Applications

Although achieving host mobility for $i3$ native applications is straight-forward, many legacy applications will remain $i3$/ROAM 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 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.

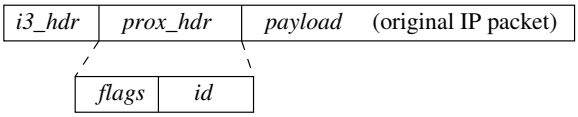


Figure 7: The format of the $i3$ packet handled by the proxy. The fields are explained in Table 1.

Figure 6 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 8 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 in $i3$ and proxy headers and sends it to CB through $i3$ using UDP. [5] The identifier of the packet is set to B's public trigger identifier, i.e., $H(B.hip)$ (see function **ip_receive** in Figure 8). The format of the packets handled by $i3$ proxies is shown in Figure 7

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 the local application. In addition, PB checks to see if the packet is addressed to its own public trigger. If it is, 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.

Assume that CB does send a packet before the timeout expires, then PB piggybacks B's local private trigger on the outgoing packet to A. Since, PB 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 private trigger to B.

[5] In order to avoid fragmentation due to the encapsulation, the maximum segment size (MSS) TCP header option in a SYN packet is decremented accordingly.

```
// 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);

// 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);
```

```
// 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);

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

Figure 8: 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 7. *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*.

When A changes its IP address from $A.cip$ to $A.cip'$ as a result of moving from one subnetwork to another, PA will insert a trigger containing the new IP address $A.cip'$ into $i3$ and remove the trigger containing the old IP address $A.cip$. 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 experiencing a high latency. The new private triggers can be exchanged 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.

6 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 [29]), called TUN[6], to transparently capture packets at user-level, and to hide host mobility from applications. The TUN 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.

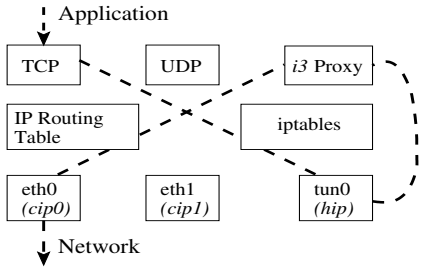


Figure 9: Data link, network, and transport layers on an end-host running the ROAM proxy software. The dashed line shows the path of an outgoing TCP packet.

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. 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 9 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 proxy

[6]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.

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 9).

6.1 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 (i.e., original IP packet, see Figure 7), the ROAM proxy suppresses duplicates during multicast-based soft handoff.

The ROAM proxy maintains a small window of MD5 [30] 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 [31]. 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 detects duplicates received on different addresses, which means that only duplicates generated by $i3$ are eliminated, and *not* duplicate packets sent by the sender (e.g., TCP dup-ack). We show in Section 7 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. Our algorithm removes 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 address (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 $1/k$ 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 $t = 30$ s.

7 Evaluation

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

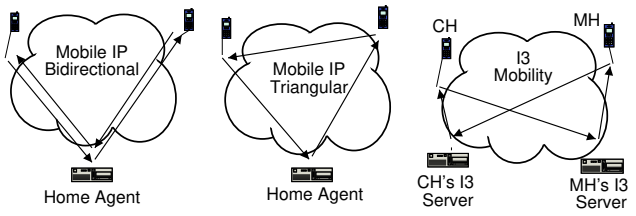


Figure 10: Routing in different mobility schemes.

7.1 Simulations

We use simulation to evaluate ROAM and Mobile IP with triangle routing, bidirectional routing. We show that 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 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.

7.1.1 Methodology

We use our own simulator to simulate $i3$ mobility and Mobile IP with its routing options. The simulator is session level, and simulates the creation, maintenance, and measurement of routes in the IP network, Mobile IP, and $i3$. We do not require the level of detail (and consequent overhead) of a packet-level simulator like NS.

Our simulation network topologies consist of three types of nodes: routers, mobility servers ($i3$ servers or HAs), and client hosts (MHs or CHs). We arrange the routers using a transit-stub topology generated with the GT-ITM topology generator [32] with 5000 nodes, where link latencies are 100 ms for intra-transit links, 10 ms for transit-stub links and 1 ms for intra-stub links. In [33], we also present simulations using a power law topology.

We define *domain* to be a group of nodes that have low latency links between them. We assume that each router forms its own domain. We consider 5000 possible client hosts, and up to 10000 mobility servers.[7] For a particular topology, we use 50 random choices from the client hosts for the home network (HN). For each choice of HN, we use 2000 random choices from the client hosts for the MH and CH, as described below.

In addition to regular IP routing, we consider three mobility routing schemes: Mobile IP with triangular routing, Mobile IP with bidirectional tunneling, and ROAM (see Figure 10).

With Mobile IP, each MH has a HA associated with it. While the HA is typically assumed to be in the HN, in practice this

[7] There is little performance improvement for more than $2N$ servers because at that point, each domain is likely to have a server.

might be hard to achieve due to deployment costs. In addition, requiring the HA to be in the HN restricts the number of MHs that can be supported. For these reasons, we assume a more incremental deployment model, where a service provider would provide one or more HAs 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. Each model defines the distance of the MH from the HN. In the uniform model, the distance of the MH from the HN is uniformly randomly selected from the interval [minimum distance, maximum distance]. 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 uniformly randomly selected from the clients. 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 10. 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 even more latency, so this comparison favors those cases over ROAM.

In all cases, we measure the 90th percentile latency stretch [8] of the various schemes.

7.1.2 *Results: Stretch vs. Infrastructure Size*

Figure 11 shows a series of graphs which compare the 90th percentile stretch of MIP with triangular routing and bidirectional tunneling ("bi"). Each graph shows a different combination of mobility model and communication model.

[8] Calculated as the ratio of the path latency using a particular mobility scheme to the shortest path latency on the underlying network topology.

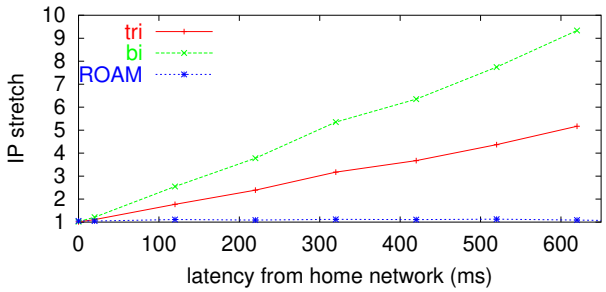


Figure 12: The 90th percentile stretch of MIP (tri/bi) and ROAM.

In the transit-stub network, ROAM matches or exceeds MIP's stretch when more than 1-2% of the transit-stub domains have a server. ROAM matches MIP when one or both of the communication end points (the MH and CH) is close to the HN. We 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 HAs increases. More HAs 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 HN. However, the figures also show that ROAM's stretch converges with MIP's when more than 50-100 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 $i3$ servers which are as close to the MH and CH as a statically configured HA.

ROAM significantly improves on MIP's stretch when neither the MH or CH are close to the HN. We expect this to be the worst case for MIP. Figures 11(a) and (c) validate this. Increasing the number of HAs in these cases does not decrease MIP's stretch because having a HA close to the HN 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 closer 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 1000 ms, while the minimum latency is only 1ms, so the impact of poor routing is very large.

7.1.3 *Results: Stretch vs. Distance from HN*

Figure 12 compares the stretch to the distance of the MH from the HN. As the distance from the HN increases, MIP's stretch increases linearly, while ROAM's stretch remains relatively constant. This simulation uses the transit-stub topol-

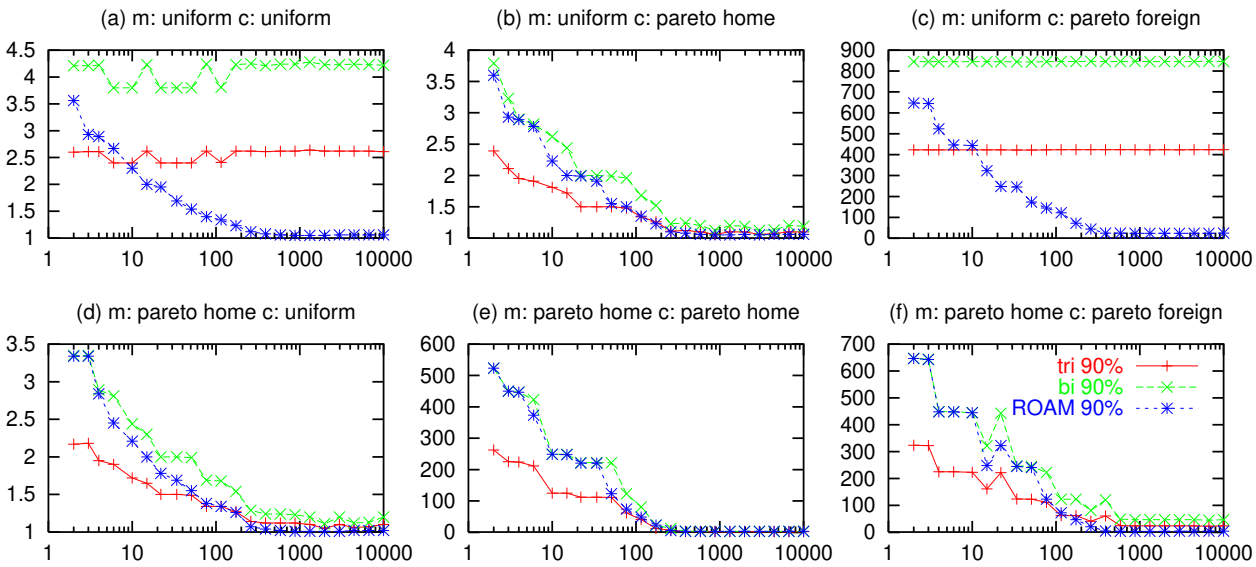


Figure 11: The 90% stretch of MIP with triangular routing ("tri"), bidirectional tunneling ("bi"), and ROAM. In all plots, x-axis represents the number of servers (home agents or $i3$ servers) on a log scale, and y-axis represents the latency stretch. Each plot corresponds to a mobility model ("m") and a communication model ("c"). "uniform", "Pareto home", and "Pareto foreign" indicate how the location is chosen, i.e., according to a uniform distribution, Pareto distribution from home network, and Pareto distribution from the foreign network.

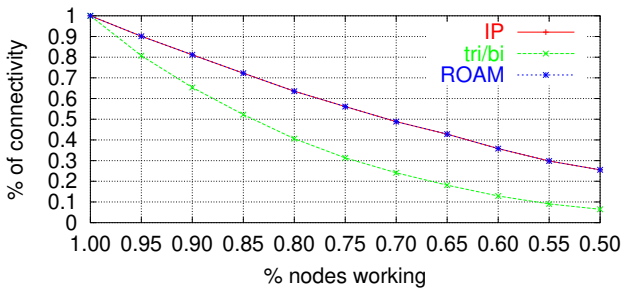


Figure 13: The robustness of IP, MIP (tri/bi), and ROAM.

ogy with 10000 mobility servers, a uniform mobility model, and a uniform communication model.

7.1.4 Results: Fault Tolerance

In addition to stretch, we also simulate node failures. We vary the failure probability of the clients and servers from 0% to 50% and perform 10,000 runs. In this simulation, we assume that both the MH and CH are mobile and have a HA. 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 HA are operational. We assume that ROAM is functional 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 a 85% likelihood of successful connectivity. When nodes have a 15% chance of failing, MIP likelihood of successful connectivity drops to only 50%. MIP is vulnerable to the failure of the HA's network connectivity. In most cases, a host has only one HA in its HN. As a result, if the HA's network connectivity fails, the MH is unreachable. In contrast, a ROAM host can use any $i3$ server in the Internet. As long as one $i3$ server is operational and the ROAM host has IP connectivity, the host is reachable.

7.2 Experiments

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

7.2.1 Results: Multicast-based Soft Handoffs

In this experiment, we perform TCP bulk transfers from the CH to the MH. The MH resides in a 10 Mbps Ethernet, and the CH and the $i3$ server reside in a 100 Mbps Ethernet. The MH initiates TCP connections from one location on its subnet, and 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 TCP bulk transfer lasting 16 seconds and we varied the number of handoffs (0, 1, 2, and 4) performed during each transfer. This was repeated ten times at each handoff frequency. Fig-

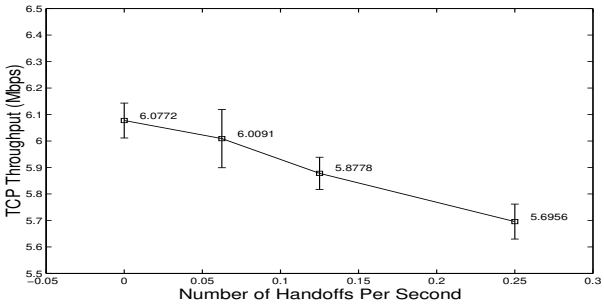


Figure 14: TCP throughput received by the MH as the handoff frequency increases. The vertical error bars show the standard deviations of the receiver throughput.

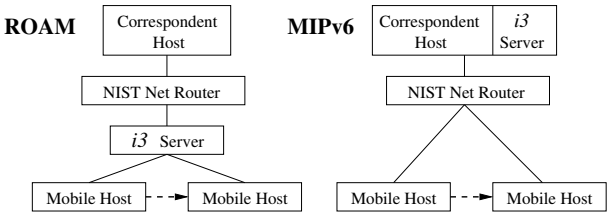


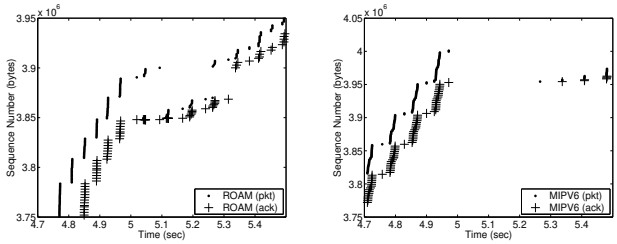
Figure 15: Network topology used for cold switch experiments. As shown by the dashed arrow, the mobile host moves from one location to another on the same subnet during handoff.

ure 14 plots TCP throughput and its standard deviation received by MH as the number of handoffs increases during the bulk transfer.

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 540 miles per hour, and cell coverage sizes with diameters of 1.5 miles. In this case, the user makes 6 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.

7.2.2 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 15 shows the experimental setup. We use the NIST Net [34] network emulation package to emulate a round trip time (RTT) of 70 ms between the MH and CH. In the setup for ROAM, RTT between the MH and the $i3$ server is approximately 3 ms. The NIST Net router delays packets between the $i3$ server and the CH by 70 ms. We emulate the MIPv6 scenario by running the $i3$ server on the same



(a) (b)

Figure 16: TCP sequence trace showing a bulk transfer with a cold-switch for (a) ROAM, and (b) MIPv6.

machine as the CH since binding updates are propagated to the CH in MIPv6. The NIST Net router delays packets between the MH and CH by 70 ms. During a cold switch, the first interface is shutdown around 35-40 ms before the second interface is brought up. During this disconnected interval, t, 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.

Figure 16 plots 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.

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.

8 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.

Eavesdropping. As discussed in Section 4.2.2, $i3$ 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.

To avoid this problem, $i3$ uses public key cryptography to protect against eavesdropping [6]. When initiating a connection, A encrypts its private trigger id_a under the public key

Routing	Header Size	Relative Overhead	Transmission Delay
IP	40B	1.25	23ms
Mobile IP	60B	1.88	28ms
ROAM	117B	3.66	41ms
ROAM w/comp	45.8B	1.43	24ms

Table 2: This table shows the total header overhead (including TCP/IP) of various routing schemes. 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.

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 $i3$. A potential disadvantage of this technique is it assumes the existence of a Public Key Infrastructure.[9]

An alternative solution would be to use an 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 do not need to have the EXCLUSIVE_ID flag set. This allows applications to use the multicast functionality via private triggers (see Section 4.2.2).

Although setting the EXCLUSIVE_ID flag ensures that no one can eavesdrop on 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. If an end-host wants to eliminate this attack, it may use again cryptography to avoid impersonation by an attacker.

Overhead. Table 2 lists the overhead of various routing schemes. 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 needs 20 additional bytes for IP in IP encapsulation. The size of $i3$ header in the current implementation is 48 bytes (of which 32 bytes is the $i3$ ID). The proxy header has a minimum size of one byte (see Figure 7). 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 IDs are piggybacked in data packets (typically only in the beginning of a connection), the overhead increases by another 32 bytes.

However, header compression can reduce packet header overhead by a factor of 5 [35]. 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.

[9] Another possibility would be to use DNS to store public keys, but then ROAM would be as secure as the DNS.

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 MH's CHs will have ROAM proxies. An alternative is to deploy a home proxy for a MH that implements the functionality of the ROAM proxy for all of its non-ROAM CHs. This home proxy is analogous to the HA in MIP 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.

9 Conclusion

In this paper we present a highly robust and efficient mobility architecture. ROAM uses an indirection infrastructure ($i3$) that gives end-hosts the ability to *control* placement of indirection points in the infrastructure. ROAM uses this ability to achieve efficient routing, fast handoff, and preserve location privacy. ROAM is as robust as the underlying IP network, and allows simultaneous mobility while not requiring

any changes to the TCP/IP protocol stack.

Simulation results show that ROAM has a low latency stretch and it is highly robust compared to Mobile IP. We evaluate a prototype of ROAM in a small testbed, and preliminary experimental results demonstrate that ROAM provides good support for soft-handoff and frequent mobility. We plan to deploy ROAM on a larger scale with end hosts and $i3$ servers spanning the continental US. In addition, we plan to explore using ROAM to compose services [6] such as transcoding and transport protocol optimization for wireless links (e.g., TCP Snoop [36]) that complement mobile routing.

References

[1] Charles Perkins, "RFC 3220: IP Mobility Support for IPv4," IETF, jan 2002.

[2] Alex C. Snoeren and Hari Balakrishnan, "An End-to-End Approach to Host Mobility," in *Proceedings of MobiCom*, 2000.

[3] N. Anerousis et. al., "TOPS: An Architecture for Telephony over Packet Networks," *IEEE JSAC in Comms*, 1999.

[4] Petros Maniatis et. al., "The Mobile People Architecture," in *ACM MC2R*, July 1999.

[5] Helen J. Wang et. al., "ICEBERG: An Internet-Core Network Architecture for Integrated Communications," *IEEE Personal Communications Magazine*, 2000.

[6] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana, "Internet Indirection Infrastructure," in *Proceedings of SIGCOMM 2002*.

[7] Mike Ritter et. al., "Mobile Connectivity Protocols and Throughput Measurements in the Ricochet MCDN System," in *Proceedings of MobiCom*, 2001.

[8] David B. Johnson and Charles Perkins, "Internet Draft: Mobility Support in IPv6," IETF, june 2002, work in progress.

[9] Stuart Cheshire and Mary Baker, "Internet Mobility 4x4," in *Proceedings of SIGCOMM*, 1996.

[10] Charlie Perkins and David B. Johnson, "Internet Draft: Route Optimization in Mobile IP," IETF, sept 2001.

[11] Pat Calhoun, Tony Johansson, and Charles Perkins, "Internet Draft: Diameter Mobile IPv4 Application," IETF, june 2002.

[12] Karim Malki et. al., "Internet Draft: Low Latency Handoffs in Mobile IPv4," IETF, june 2002, work in progress.

[13] Alper Yegin et. al, "Internet Draft: Fast Handovers for Mobile IPv6," IETF, march 2002, work in progress.

[14] Eva Gustafsson, Annika Jonsson, and Charles E. Perkins, "Internet Draft: Mobile IP Regionalized Tunnel Management," IETF, august 1999, work in progress.

[15] Hesham Soliman, Claude Castelluccia, Karim El-Malki, and Ludovic Bellier, "Internet Draft: Hierarchical MIPv6 Mobility Management," IETF, july 2002, work in progress.

[16] "Host Identity Protocol," `http://homebase.htt-consult.com/hip.html`.

[17] "LIN for IPv6," `http://www.lin6.net/`.

[18] Jerome Saltzer, "RFC 1498: On the Naming and Binding of Network Destinations," aug 1993.

[19] Faramak Vakil et. al., "Internet Draft: Supporting Mobility for TCP with SIP," IETF, december 2000, work in progress.

[20] Jayanth Mysore and Vaduvur Bharghavan, "A New Multicasting-Based Architecture for Internet Host Mobility," in *Proceedings of ACM/IEEE MobiCom*, 1997.

[21] J. Mysore and B. Vaduvur, "Performance of Transport Protocols Over a Multicasting-based Architecture for Internet Host Mobility," in *IEEE ICC*, 1998.

[22] S. Deering and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM TOCS*, May 1990.

[23] Ahmed Helmy, "A Multicast-based Protocol for IP Mobility Support," in *Proceedings of NGC*, 2000.

[24] Ahmed Helmy and Muhammad Jaseemuddin, "Efficient Micro-Mobility using Intra-domain Multicast-based Mechanisms (M&M)," Tech. Rep., USC, aug 2001.

[25] Frank Dabek, Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, "Wide-area Cooperative Storage with CFS," in *Proc. ACM SOSP'01*, Banff, Canada, 2001, pp. 202–215.

[26] Diane Tang and Mary Baker, "Analysis of a Metropolitan-Area Wireless Network," in *Proceedings of MobiCom*, 1999.

[27] "Wireless LAN Medium Access Control and Physical Layer Specifications," Tech. Rep., IEEE Computer Society, 1999.

[28] Mark Corner and Brian Noble, "Zero-Interaction Authentication," in *Proceedings of MOBICOM 2002*.

[29] Mary G. Baker, Xinhua Zhao, Stuart Cheshire, and Jonathan Stone, "Supporting Mobility in MosquitoNet," in *Proceedings of USENIX Technical Conference*, 1996.

[30] R. Rivest, "RFC 1321: The MD5 Message-Digest Algorithm," IETF, apr 1992.

[31] Alex C. Snoeren et. al., "Hash-Based IP Traceback," in *Proceedings of SIGCOMM*, 2001.

[32] "GT-ITM," `http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html`.

[33] Shelley Zhuang, Kevin Lai, Ion Stoica, Randy Katz, and Scott Shenker, "Host Mobility Using an Internet Indirection Infrastructure," Tech. Rep., U.C. Berkeley, June 2002.

[34] "NIST Net Network Emulator," `http://snad.ncsl.nist.gov/itg/nistnet/index.html`.

[35] Jeremy Lilley, Jason Yang, Hari Balakrishnan, and Srinivasan Seshan, "A Unified Header Compression Framework for Low-Bandwidth Links," in *Proceedings of MobiCOM*, 2000.

[36] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz, "Improving TCP/IP Performance over Wireless Networks," in *Proceedings of MOBICOM 1995*.