A Simple Solution to Scale-Free Internet Host Mobility

J.J. Garcia-Luna-Aceves and Spencer Sevilla
{ jj, spencer }@soe.ucsc.edu
UC Santa Cruz, Santa Cruz, CA

Abstract—**We introduce a simple solution for the support of host mobility in the Internet called DIME (Dynamic Internet Mobility for End-Systems). DIME is based on dynamic address translation between the transport and network layers of end hosts, combined with a new out-of-band protocol that updates host-address bindings between communicating hosts opportunistically. It does not require modifications to the end-host operating systems, end-user applications, existing communication protocols or hardware, or the domain name system and any host-identifier namespace. A number of experiments based on a Linux daemon implementation of DIME are used to show that DIME is deployable on a wide range of hardware, and that it outperforms existing mobility proposals such as MIPv6 and HIP across a wide range of performance metrics.**

I. Introduction

The development of the Internet architecture [1] was motivated by the advent of local-area networks (LAN), satellite-based networks, and packet-radio networks that required interconnection with one another. Although mobility has always been an inherent aspect of packet-radio networks, mobility was assumed to take place within a single packet-switching subnet, and to date the basic Internet does not support host mobility seamlessly. Connections at the transport layer must be restarted when a host changes network attachment points, and cannot take advantage of multiple network paths simultaneously. Within the specific context of the Internet protocol stack, this limitation is the result of transport-level protocols (e.g., TCP and UDP) using IP (Internet Protocol) addresses to consistently *identify* communicating processes in remote hosts, while the network layer uses IP addresses to *locate* host endpoints in the network. For communication between two communicating hosts to persist across address changes, these two functions must be separated. Surprisingly, no efficient solutions have been proposed to date to solve what is called the *identifier-locator split* problem.

Supporting host mobility in the Internet and solving the identifier-locator split problem has become critical for the future of the Internet. The number of mobile-only users has already overtaken the number of desktop-only users [2], and smartphone subscriptions are predicted to hit an expected total of 26 *billion* mobile devices [3] in 2020. Section II summarizes the prior work on Internet mobility support, and the attempts that have been made to solve or eliminate the identifier-locator split problem of the Internet architecture. The solutions to date range from using network-level mechanisms that cope with mobility in the forwarding plane to doing away with addresses altogether and using names as identifiers that denote "what" rather than "where." The common characteristic in all of this prior work, including information-centric network (ICN) architectures that attempt to replace the current IP Internet, is that the identifiers used in the data plane of communication protocols are the same as those used within the protocol stacks of communicating entities.

The contribution of this paper is the description of a simple approach for the support of Internet host mobility that is completely seamless with respect to applications, the Internet routing infrastructure, the domain name system (DNS), and the data plane of existing communication protocols. Accordingly, it can be deployed in an incremental manner on top of an unmodified operating system *without* requiring any changes to intermediate network devices, applications, or the system kernel. We call this new approach *Dynamic Internet Mobility for End-Systems* (**DIME**).

Section III describes DIME, which consists of two main architectural components. A modified form of network address translation (NAT) that we call *StackTrans* is inserted between the transport and network layers of the protocol stack, such that the transport layers and above see unmodified network addresses (i.e., identifiers), and yet datagrams are addressed to the current network address of a host (i.e., locators). The address bindings used by StackTrans as hosts gain and lose network addresses are updated by a simple out-of-band end-to-end signaling protocol that we call the *Internet Host Mobility Protocol* (IHMP).

Section IV describes a concrete Linux implementation of DIME and presents detailed comparisons with Mobile IP [4] and the Host Identity Protocol [5]. Our results show that DIME is much more lightweight and deployable, runs on a wider range of hardware and software, and is significantly more scalable across a wide range of performance metrics. Section V concludes the paper.

II. Related Work

There exists a vast body of work addressing host mobility in the Internet [6], [7], [8], [9], [10], [11], [4], [12], [13]. The many solutions that have been proposed in the past either require modifications to the protocols used in the data plane at the network, transport, or application layers of the existing Internet architecture, or advocate replacing the Internet architecture altogether.

A. Network-Layer Support

The first proposals to address host mobility in the Internet, including Mobile IP [14], [15], [16], [17], do so entirely within the network layer. The intended benefit of this approach is that by restricting all alterations to the network layer, higher-layer protocols (i.e., TCP and application-layer protocols) can take full advantage of any such mobility solutions without modification. Unfortunately, all network-layer approaches to host-mobility support have two major limitations.

In the Internet architecture, processes at the transport and application layers bind IP addresses to identify the hosts where the intended communicating process are located; it follows that this binding between a host and its IP address must be preserved to avoid disrupting communication. Solutions proposed to date maintain this binding either by updating routing tables as a host moves through the Internet [16], [17], or by dividing the network address or address-space into *identifiers* and *locators* [14], [15]. Unfortunately, updating routing tables as hosts move incurs an untenable amount of network control signaling and increases the size of routing tables, and using special addresses as identifiers significantly fragments the identifier address-space. In addition, network-layer mobility support requires significant coordination, standardization, and infrastructure investment, because the proposed schemes necessarily alter network-layer protocols in the control plane *and* data plane. Such an effort is impractical, because it requires replacing large portions of the routing infrastructure within the deployment domain, and also requires a globally-coordinated time for migration.

B. Transport-Layer Support

A number of approaches [6], [18], [19], [20], [21], [22], [23], [24], [25] support mobility at *end-hosts*, typically enacted as a part of the transport layer. The transport protocol uses additional signaling or options to update the IP address bound to a connection as a host experiences network-address mobility.

Even though existing proposals for mobility support at the transport layer do not require changes to the routing infrastructure, they alter transport protocols (e.g., TCP) and introduce incompatibilities with NAT boxes. Furthermore, because IP address migration is instantiated on a per-connection basis, in the proposed schemes, these schemes suffer from scalability problems.

Another concern with these approaches is that the transport layer is required to assume a specific mobility model (e.g., "always/sometimes/never migrate this connection across hosts") and apply the same model to every single application running in the system. This limits the usefulness of the approach.

C. Application-Layer Support

The main goals of supporting host mobility at the application layer are to eliminate the need for changes in the routing infrastructure and to provide applications the flexibility to handle mobility differently from other applications.

A number of proposals [26], [27], [28], [29] use the Session Initiation Protocol (SIP) to enable applications to uniquely identify, join, and leave communication sessions using a "user@host" [27] identifier. The limitations of these proposals are that they require a new API to be presented to applications, focus their efforts on the real-time session abstraction, and generally do not support the reliable data-stream paradigm leveraged by all TCP-based applications.

From an architectural point of view, SIP-based solutions are essentially mobile IP at the application-layer, given that they rely on application-layer home agents to provide redirection when users or end-hosts are mobile. SAMP [30] addresses the last point by maintaining session information in an overlay network; however, SAMP incurs the same costs associated with distributed hash tables (DHTs), such as latency and control message churn.

Many other works [31], [32], [25], [33], [34] propose new application interfaces to better support host mobility. However, none of these works change the binding or socket abstractions presented to applications. They simply provide a shim library wherein applications identify connections by the *originally* bound {IP, port} tuple, and the library responds to mobility events by opening a new socket to the new identifying tuple.

D. New Internet Architectures

Several proposals have been advanced to redesign the current Internet architecture to address such issues as information-centric usage patterns, privacy and security, evolvability, and of course mobility. All these proposals attempt to solve the early-binding problem of the IP Internet, and include identifier mobility as an integral aspect of their designs.

Many of these architectures [35], [36], [5], [37], [38], [39], [40], [41] introduce one or more new identifying layers into the existing protocol stack in the Internet as a way to resolve the naming and addressing issues of the IP Internet. These layers typically include a host- or endpoint-identity layer based on either the DNS [42], [25] or cryptographic host identifiers [5], [35], [37], [41], [33], and can also include a layer used to identify individual application services [38], [40], [39], [35].

A few information-centric networking (ICN) architectures (e.g., NDN [43]) advocate a clean-slate redesign of the entire control and data planes of the Internet focusing on name-based content forwarding and caching. regarding mobility, the goal is to eliminate the identifier-locator split problem altogether by eliminating addresses. However, this requires the introduction of a new global namespace and forwarding information bases that are much larger than traditional forwarding tables, and does not result in more efficient packet forwarding than using addresses [44].

E. Limitations of Prior Solutions

All the proposed alternatives to the Internet architecture suffer from two main drawbacks. First, they require introducing a new identifying layer that requires the redesign of network applications, operating systems, communication protocols in the data plane, the directory services used in networks, middleboxes, or routers. Second, if adopted, any of these proposals would essentially "lock in" a new set of identifiers for content, services or devices as part of the new Internet architecture, which is a major drawback. Just as the original design by Cerf and Kahn [1] could not predict the constraints and challenges of today's networks, we argue that current proposals cannot accurately envision all the naming challenges of the future Internet.

The fact that so many proposals addressing the Internet host mobility problem have failed to provide a simple solution that does not require any changes to end-user applications, the communication protocols operating in the data plane, the DNS, or the routing infrastructure may appear to indicate that no such as solution is possible. Indeed, part of the justification of recent ICN architectures is the argument that using addresses is inherently limiting in supporting the mobility of services or content. However, as we have argued recently [45], [46], [47], the limitations of prior proposals stem from the assumption that the identifiers used in the communication protocols operating in the data plane must be the same as the identifiers used within the protocol stack of a host or a router to pass information across layers of the stack. We have demonstrated that this does not have to be the case and that indirection within the protocol stack can be done efficiently to allow names and addresses used in communication protocols to differ from the identifiers used inside hosts and router to refer to resources, connections, or remote processes.

In addition, a review of the prior work also reveals that most if not all prior proposals on the resolution of names to addresses assume that either: (a) a directory service maintains the mappings from a name to a set of addresses and that a host must obtain all the mapping from the directory service directly or be assisted by the routing infrastructure for redirection; or (b) the routing infrastructure operates directly on names and hence maps names to routes directly.

Our design of DIME, which we present in the next section, is motivated by two simple observations. First, our prior work on "hidden identifiers" in the protocol stack [45], [46], [47] demonstrates that hosts and routers can denote resources, connections and remote processes with identifiers that need not be the same as those used as part of communication protocols. Second, updating the mappings from names to addresses need not be done solely through a common directory service, and indeed communicating hosts or routers can update one another directly if done properly.

III. DIME

Figure 1 illustrates the functional architecture of DIME, which is comprised of two independent components. DIME introduces an address redirection module in the data plane that we call *StackTrans*, and uses a lightweight out-of-band signaling protocol called the Internet Host Mobility Protocol (IHMP) in the control plane to update the identifiers used by StackTrans.

A. StackTrans and Redirection in The Data Plane

The goal of StackTrans is to dynamically readdress datagrams as they pass between the transport and network layers of the stack, such that the application and transport layers use an unchanging IP address for a host (i.e., an identifier) and the network layer uses the current network address for a host (i.e., a locator). StackTrans achieves this split through the two-step address translation process illustrated in Figure 2. The IP address is first bound by a socket to an entity called a *Host Identifier* (HID), and then the HID is mapped to its current network address.

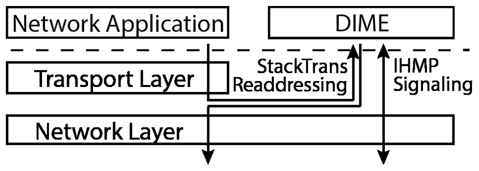

Fig. 1. DIME System Architecture

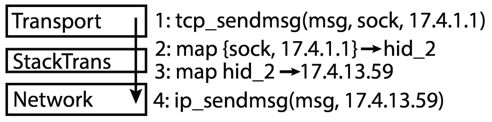
Transport
StackTrans
Network
1: tcp_sendmsg(msg, sock, 17.4.1.1)
2: map {sock, 17.4.1.1}→hid_2
3: map hid_2→17.4.13.59
4: ip_sendmsg(msg, 17.4.13.59)

Fig. 2. StackTrans Packet Readdressing

The benefit of this design is that it is the only way to achieve a true identifier-locator split without requiring any changes to transport-layer protocols or inserting a new layer in the protocol stack. The first translation (Steps 1 and 2 of Figure 2) enables the use of an unmodified host kernel by allowing application and transport protocols to bind an unchanging IP address across the course of a connection, just as they do today. Once multiplexed to a HID, the HID table provides a unifying location in the system to store and maintain all address information for a specific host: similar to the design of routing tables today, any modifications need only be entered into the HID table to immediately impact every connection in the system. The second translation (Steps 3 and 4 of Figure 2) enables the use of an unmodified routing infrastructure by ensuring that datagrams are addressed to the correct network address for a host before even reaching the host's network layer for forwarding.

StackTrans manages the first translation via a *Socket Table*, illustrated in Figure 3(a), which maps locally-bound foreign addresses to HIDs. When a socket sends data to a new foreign address via `connect` or `sendto`, the address is looked up in the HID Table and translated to an existing HID; if no such HID exists, a new one is created. Once the HID is obtained or generated, the {socket, daddr, hid} tuple is stored in the socket table, and used to ensure that (1) all communication from the socket to the destination address is mapped to the correct HID and (2) all communication from the HID to the socket is mapped back to its initial address. Socket Table mappings must be kept separately for each socket because it is possible for multiple sockets to use different network addresses to refer to the same HID if the foreign host experiences multiple address-changes (e.g., socket $s1$ opens a connection to address $a1$, the host moves to address $a2$, and then socket $s2$ opens a connection to address $a2$).

(a)

Socket	Bound Address	HID
s1	17.4.1.1	hid_2
	192.168.54.1	hid_1
s2	192.168.52.10	hid_1
	17.4.1.1	hid_2

(b)

HID	Active	Unreachable	Local	Key
hid_1	192.168.54.1 13.43.56.7	NULL	192.168.54.8	key1
hid_2	17.4.13.59	10.0.0.8	17.4.13.3	NULL

Fig. 3. Socket and Host Identifier Tables of StackTrans

A HID is an internally-kept value that semantically refers to a foreign host. Host identifiers are not new, but prior works implement these identifiers using a new global namespace and layer in the network stack. This is a crucial limitation of prior work, because it requires major changes to the network stack and transport layers. By contrast, the HIDs used in DIME are simply indices into the HID Table: HIDs have no external value, are not bound by higher layers, and are never propagated over the network.

Using HIDs in DIME provides a unifying location to integrate mobility signaling into the data plane. By multiplexing {socket, daddr} tuples to a HID, rather than directly to an address, DIME achieves an abstract split that enables multiple addresses to be kept for a host. Additionally, given that multiple {socket, daddr} tuples can be multiplexed to the same HID, DIME enables a single control message exchange (indicating an address handoff, for example) to be immediately reflected at every connection to the HID at once. Finally, since the only function of a HID in the data plane is to be multiplexed to an address, we achieve these benefits *without* altering transport protocols, adding a layer to the stack, or even defining a new namespace for host identifiers.

Figure 3(b) illustrates that each entry in the HID Table stores three separate sets of addresses: *active* addresses, *unreachable* addresses, and *local* addresses. Active addresses are addresses currently owned by a foreign host that are reachable by the local host. Unreachable addresses are addresses currently owned by the foreign host that the local host is currently unable to contact, but *may* become reachable later on. Local addresses are those addresses owned by the local host that are reachable by the foreign host; this set is identical to the set of active addresses in the foreign host's HID table. The HID of a host is also optionally bound to a *host key* for the host.

The reason for storing the active addresses for a host is obvious. The other fields in the HID table support the signaling of the proposed IHMP. Local addresses are stored so that when a host sends IHMP messages, it is able to identify itself with an address that is ensured to be in the HID table of the foreign host. Unreachable addresses for a host are stored to optimize IHMP signaling around address up, down, and handoff events. Finally, the host key is used for authenticating IHMP messages from the host, and is explicitly *not* used for data-plane communication.

An important decision in the design of DIME was to make it independent of the directory services used in the Internet today or in the future, i.e., the domain name system (DNS) as of today. Accordingly, end-user applications resolve domain names to addresses through their name resolvers without any modifications. StackTrans essentially "eavesdrops" on transport-level traffic (TCP connections or UDP transactions) by using the socket table to record the IP address bound by a specific socket and ensure that all communication to and from that socket is multiplexed to the original IP address. Hence, when the first packet for a UDP or TCP flow is sent, StackTrans creates an entry in the Socket and Host Identifier tables stating the active IP address and HID for the socket. In turn, the creation of new entries in the StackTrans tables prompts the IHMP daemon to start sending messages to the remote host associated with the new HID and active IP address.

B. *IHMP and Updating Addresses in The Control Plane*

The address indirection provided by StackTrans allows remote processes and connections to be readdressed at end hosts without any modifications to the data plane of the protocol stack. This enables the signaling needed to update host addresses as hosts move to be taken out of the data plane and enact it as a simple end-to-end signaling protocol, which we call IHMP.

IHMP listens for network address events at the local host (e.g., a network interface going up or down) and communicates these changes to foreign hosts via UDP messages. IHMP messages are sent over UDP to limit the overhead they incur and to allow IHMP daemons to interpret dropped messages as a sign that a path may not be sufficiently reliable. Using UDP instead of ICMP enables NAT detection and traversal. When an IHMP daemon receives a message from a foreign host, it updates the HID Table to reflect the changes, at which point StackTrans immediately incorporates them into the data path.

Figure 4 illustrates the IHMP message format. The control field states the message type, which can be a HELLO, a path probe, an address event (e.g., new address, address unreachable, address lost), a handoff, or an acknowledgment from a host or router. The Sequence Number is a randomly-seeded 16-bit value incremented with each message and echoed by every responding `ACK`. The sender *may* elect to append a digital signature to the message, but *must* identify itself to the receiver via a local IP address in the Host ID field.

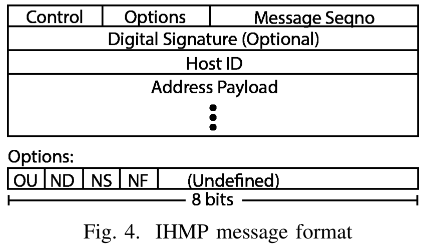

Fig. 4. IHMP message format

The Host ID field in IHMP messages differs from all other proposals [14], [15], [5], [42], [25] that use end-to-end updates. Whereas all prior proposals rely on a separate host identifier namespace (e.g., a DNS hostname) to consistently identify a host across network address changes, IHMP is specifically designed *not* to rely on or assume any such namespace. This makes IHMP much more flexible, modular, lightweight, and deployable. However, it also means that the only method by which a receiving host can identify the source of an IHMP message is by its IP address, which is stored in the Host ID field. A sender populates the Host ID field with an IP address that will multiplex to the sender's HID at the receiver. This multiplexing is ensured by choosing an IP address from the set of local addresses bound to the receiver's HID, and is the reason for storing the local addresses reachable by a foreign host.

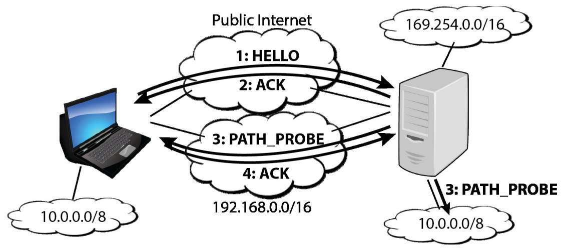

Fig. 5. HELLO message exchange

1) Hello Exchange: When StackTrans at an active host creates a new HID table entry, it prompts IHMP to query the foreign host for the presence of other network addresses, and advertise its set of local addresses to the foreign host. This is done via a simple two-way `HELLO` message exchange, illustrated in Steps 1 and 2 of Figure 5. The first `HELLO` message contains all the network addresses the active host wishes to advertise to the foreign host, including the source network address used to transmit the `HELLO` message.

When the foreign host receives a `HELLO` message, it creates an entry in its HID table for the active host, adds the source address of the message to the HID's set of active addresses, the destination address of the message to the set of local addresses, and every other address in the `HELLO` message to the set of unreachable addresses. Finally, it replies with an `ACK` to the address used by the active host to send the `HELLO` message. Similar to the `HELLO` from the active host, this `ACK` contains all addresses the passive host wishes to advertise to the active host. Upon receipt of the `ACK`, the active host adds the destination address to the set of local addresses of the HID, and all other advertised addresses to the set of unreachable addresses of the HID.

From the perspective of the foreign host, the other addresses included in the `HELLO` message from th active host are either: (a) reachable by the host (e.g., a publicly reachable IP address); (b) unreachable by the host (e.g., in a private network the host does not have an address in); or (c) *potentially* reachable (e.g., in a private network that the host has an address in, but is not necessarily the same private network). The passive host is responsible for discovering which advertised addresses are actually reachable, and accomplishes this by iterating through the set of addresses provided by the active host and sending a `PATH_PROBE` message to each address that falls under case (c); this process is illustrated in Step 3 of Figure 5. Note that the active host initially reaches the passive host over the public Internet, and indicates (via the `HELLO` message) that it also has addresses in the 192.168.0.0/16 and 10.0.0.0/8 networks. As a result, the passive host sends `PATH_PROBE` messages out over the 192.168.0.0/16 and 10.0.0.0/8 networks to which it is connected, but does *not* send a `PATH_PROBE` message over the 169.254.0.0/16 network.

When the active host receives a `PATH_PROBE` message, it moves the source address to the set of active addresses of the HID, adds the destination address to the set of local addresses of the HID, and replies with an ACK bound to the same address set (Step 4 of Figure 5). Upon receiving the `ACK`, the passive host performs the same operation. This exchange ensures that after completion, both hosts have a symmetric HID table in terms of local and active addresses.

2) Address-Up Events: When a host obtains a new network address, it is responsible for communicating this new network address to all foreign hosts in its HID table. Just as with path probing, the IHMP signaling takes one of three forms, depending on whether the foreign host is known reachable, known unreachable, or potentially reachable.

If a foreign host is known reachable, the active host sends an `ADDR_UP` message to the foreign host. The active host sends the message from its new address, and *also* explicitly encodes this address in the IHMP message.[1] The Host ID field of an `ADDR_UP` message is populated by any of the local addresses that have successfully been advertised to the passive host

If a foreign host is known unreachable, the active host sends an `ADDR_UP_UNREACHABLE` message to the foreign host. As above, the message's Host ID field is populated by a local address advertised to the passive host, and the message explicitly encodes the new network address in the message payload. However, since it has already been determined that the foreign host is not reachable by the new network address, the active host transmits the message over any network address of the foreign host available to it.

If a foreign host is potentially reachable, the active host must determine whether the foreign host is reachable by this new interface. To accomplish this, the active host creates an `ADDR_UP` message as if the foreign host is known reachable, but effectively uses it similarly to the `PATH_PROBE` message: after sending the message, it waits for an `ACK`. If the active host does not receive an `ACK` in an appropriate amount of time, it determines that the foreign host is *not* reachable over this interface, and reverts to the `ADDR_UP_UNREACHABLE` exchange described for the case when a foreign host is known unreachable.

In all of these cases, when the foreign host receives an `ADDR_UP` or `ADDR_UP_UNREACHABLE` message, it adds the corresponding address to the active or unreachable set

[1]This explicit address encoding allows us to detect and mitigate NATs.

of addresses for the HID, updates its local address set if appropriate, and replies to the active host with an ACK.

3) Address-Down Event: When a host loses a network interface and address, it checks the local address set for each HID and partitions its set of foreign hosts into hosts that are still reachable and hosts that are no longer reachable.

If a foreign host is no longer reachable (i.e., the lost network address was the only network address that could reach the foreign host), then the HID table entry is removed and an error message is sent to any network application that had bound the corresponding HID.

For all hosts that are still reachable, the active host inspects the HID's set of local addresses to determine if the foreign host was able to communicate with the lost network address or not. If the foreign host was reachable by the lost address, then the active host removes the lost address from the set of local addresses and sends an ADDR_DOWN message to the foreign host. If the host was *not* reachable by the lost address, then the active host simply sends an ADDR_DOWN_UNREACHABLE message. In both cases, the Host ID field contains an address that the active host is still reachable on, and the payload of the message contains the lost address. When the passive host receives the message, it updates its own address-sets accordingly and responds with an ACK.

4) Handoff Event: A handoff event occurs when a host loses one network address and gains another network address. These network addresses do *not* have to be bound to the same network interface, because there is no difference from the perspective of the network layer and above - either way, reachability may have changed, and the address-change must be discovered and synchronized between end hosts.

As with address-up events, an active host starts the handoff process by classifying all foreign hosts as known reachable, known unreachable, or potentially reachable with respect to the new network address. The active host sends HANDOFF messages to known-reachable hosts, HANDOFF_UNREACHABLE messages to hosts known to be unreachable, and uses HANDOFF messages to probe network reachability when it is unknown. In all cases, the active host sends the message from the new network address, encodes both new and old network addresses in the payload, and selects any address from the set of local addresses for the Host ID. When the passive host receives such a message, it updates its HID table accordingly and replies with an ACK.

The handoff process must incorporate a small amount of extra logic compared to the address-up event, because the host is also *losing* a network address. This address loss means that the active host might be completely disconnected from some foreign hosts, and this lost address *must* be communicated to all foreign hosts.

For clarity, the handoff message uses the bit OU (OLDADDR_UNREACHABLE) in the options field to indicate whether the *old* address was reachable by the passive host or not; this bit effectively represents the difference between the ADDR_DOWN and ADDR_DOWN_UNREACHABLE messages in Section III-B3.

C. Security of IHMP

Spoofed IHMP messages have the potential to disrupt and redirect communication. Accordingly, IHMP must provide a mechanism for ensuring message integrity. Given that IHMP messages are self-contained, transmitted out-of-band, and not bound to any one existing communication session, we elect to use a public/private key digital signature model as opposed to ensuring the confidentiality of the message via an encrypted end-to-end channel.

IHMP validates incoming messages by obtaining a public key in various ways, binding this public key to a HID, and using this key to verify the digital signature of any messages subsequently received from this HID. If the public key is bound to a host identifier (and therefore stored in an identifier namespace), then it can be resolved when the identifier is mapped to a set of locators. Additionally, the public key could also be obtained by alternate out-of-band processes, the HELLO message exchange, or *a priori* key information - for example, a network administrator may simply install a set of public keys onto client laptops and specify that a certain key is always used for a specific IP address.

For outgoing messages, IHMP enables a private key to be bound to either all addresses ($*$) or a specific list of addresses or subnets with the command set_local_key. The IHMP daemon stores this key and signs outgoing IHMP messages as appropriate. While host-key bindings are generally expected to be long-lived [5], [48], these private keys used to sign messages may be changed or updated with subsequent calls to set_local_key. It is then considered the responsibility of the administrator or entity changing the key-pair to publish the public key as appropriate.

D. NAT Traversal for IHMP

Despite the fact that network address translators (NATs) are the primary violators of the end-to-end argument [49] today, NATs are generally expected to be "here to stay" for the foreseeable future [50], [51]. IHMP flags NAT addresses separately, and stores them for a HID as a nat_addr:host_addr tuple.

IHMP-aware NATs indicate their presence by setting a NS (NAT_SUPPORTED) bit in all IHMP messages that traverse the NAT, and send a NAT_ENTRY message to the non-NATed host as they create new port-mappings for each connection as illustrated in Figure 6. This mapping is stored at the end-host and used to map the NATed port back to the original source port for delivery.

IHMP detects IHMP-unaware NATs by checking the source IP address of the message against the IP address encoded in the message payload, and indicates this via a ND (NAT_DETECTED) flag in the ACK. Since IHMP cannot migrate connections into IHMP-unaware NATs, it stores them as unreachable addresses unless a connection must be initiated behind a NAT. This case is indicated via a NF (NAT_FORCED) flag in the initial HELLO exchange, at which point the HID is created separately as a one-to-one mapping and no further IHMP signaling occurs.

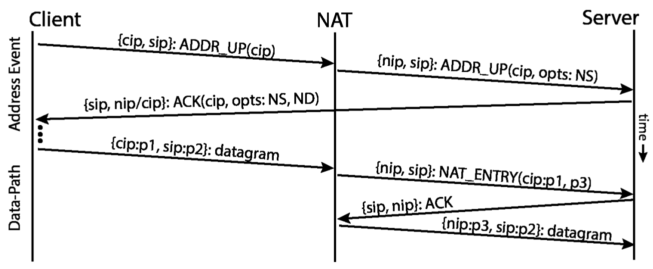
Client
NAT
Server
Address Event
Data-Path
time
{cip, sip}: ADDR_UP(cip)
{nip, sip}: ADDR_UP(cip, opts: NS)
{sip, nip/cip}: ACK(cip, opts: NS, ND)
{cip:p1, sip:p2}: datagram
{nip, sip}: NAT_ENTRY(cip:p1, p3)
{sip, nip}: ACK
{nip:p3, sip:p2}: datagram

Fig. 6. Mobility signaling with NAT

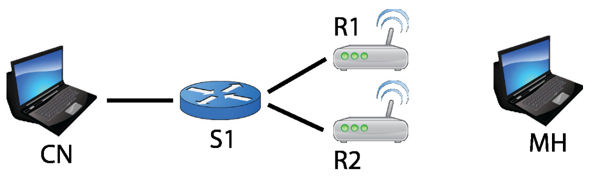

Fig. 7. Testbed topology

IV. Implementation and Evaluation

We implemented DIME as a user-space daemon and Loadable Kernel Module (LKM), deployed our code on two laptops running Ubuntu, the *Mobile Host* (MH) and the *Corresponding Host* (CH), and connected them with a switch and two routers to create the topology shown in Figure 7. In the topology, each node has a globally-reachable address, each router advertises a different subnet, and we used the `netem` utility to introduce 60ms of latency on all traffic that flows across the switch [52], [53].

To provide more consistent results and remove variance, we induced address up and down events programmatically via the `ip` command, with a five-second gap between each event; we also conducted each experiment ten times and provide mean values.

We present the results obtained with DIME and compare it with Mobile IP (MIPv6) and the Host Identity Protocol (HIP), which we chose as "flagship" examples of mobility support for Internet hosts. The performance metrics we use are the data-plane throughout, the size of socket tables, and the size of the implementation.We conducted a standard mobility experiment in which the Mobile Host moves from R1 to R2 while conducting a throughput test to the Corresponding Host. We compare DIME against MIPv6, HIP, and multipoint TCP elsewhere [54], where we also discuss signaling latencies and the number of signaling packets sent after handoffs.

A. Deployment and Configuration

Table I provides a summary of the effort required to configure each protocol even for the basic testbed in Figure 7. MIPv6 stands out by far as the most fragile and ossified approach. MIPv6's reliance on deep kernel integration requires a custom kernel and a user-space daemon at the end hosts. However, both codebases have been abandoned for several years, do not support 64-bit architectures, and are no longer compatible with any current Linux distribution. Additionally, MIPv6 was the only protocol to require a purely IPv6 testbed as well as multiple daemons (`mip6d`, `radvd`, and `hostapd`) running on routers R1 and R2 *and* the end-hosts.

Requirement	MIPv6	HIP	DIME
Daemons	4	1	1
Config. Files	3	2	1
App Mods.		✓	
System Configs	✓		
Custom Kernel	✓		
Router Mods.	✓		

TABLE I
Deployment Requirements

Configuring HIP relies heavily on manually encoding static, preconfigured bindings at both end hosts,[2] and HIP's use of the 1.0.0.0/8 block for LSI bindings raised questions about support for users that do not want to memorize IP addresses or applications that insert the address into the protocol itself (e.g., FTP).

By contrast, DIME "simply works" and exists as a single, standalone user-space daemon, requiring no configurations or modifications to application binaries or the underlying OS. DIME's leverage of existing IP address bindings enables it to work without the need for specific namespace bindings or "pseudo" IP addresses, and DIME's use of translation instead of encapsulation makes it the only approach that dynamically supports preexisting connections that were established before the DIME daemon was operational.

B. Data Plane Throughput

Since a mobility solution must not negatively impact the data-plane, Figure 8 provides the TCP goodput seen over a fifteen second throughput test, over both a soft handoff (the new address-up event happens five seconds before the address-down event) and hard handoff (the reverse order of operations). We examine goodput instead of throughput to provide a single unifying metric that accurately reflects the performance seen by network applications, and present TCP results. UDP goodput results collected for DIME, MIPv6, and HIP were all similar to the shown results shown for TCP.

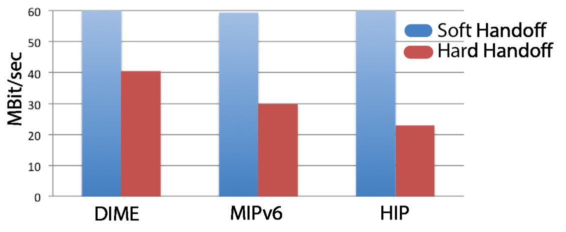

Fig. 8. TCP Handoff Goodput

Figure 8 shows that each protocol provides almost identical results over a soft handoff. This is because despite the different architectures and handoff signaling needed, each protocol has enough time to compete the handover signaling and maintain

[2]HIP architects have advocated using the DNS to dynamically resolve HIT bindings; the lack of such support highlights the importance of R3.

a constant data-rate (bound by the physical links) when the first address is lost.

Since the disconnection lasts five seconds and the total transfer lasts fifteen, it is intuitive that the maximum possible goodput for the hard handoff scenario will be approximately 2/3 of the soft handoff. Figure 8 shows that DIME achieves this value almost exactly, and noticeably outperforms all other proposals; we attribute this to a combination of DIME's faster control message exchange and transparency with respect to TCP. MIPv6 migrates the existing TCP connection without triggering any congestion-control backoff, but suffers from a noticeably longer handoff procedure. HIP's large decrease in goodput appears to be the result of "buffer bloat" at the HIP daemon during the disconnection, which in turn created erratic and unfavorable interactions with TCP's congestion control algorithms for the remainder of the connection.

C. Socket-Table Scalability

One of the defining characteristics of StackTrans is its use of *two* translation tables, instead of one. This raises questions of scalability at end-hosts, since the socket table scales CPU usage with the number of open sockets in the system. We explored how the size of the socket table affects data-plane throughput by clearing the socket table, adding n "fake" socket-table entries at the mobile host, and then starting a UDP throughput test from the mobile host to the corresponding host. In the interest of evaluating performance on a resource-constrained platform most likely to be bottlenecked by CPU usage, we ran this test on a RaspberryPi Model B+.

Figure 9 provides our results, which show that throughput is completely unaffected by socket-table size until approximately 1500 open entries, at which point it starts degrading almost linearly. Though these results show that a large socket table eventually does degrade performance, they still reflect exceedingly well on StackTrans when considered in a greater context. Specifically, the target hardware platform is not intended for large-scale serving, and is likely to hit other performance bottlenecks if it were to actually support this many simultaneous open network connections.

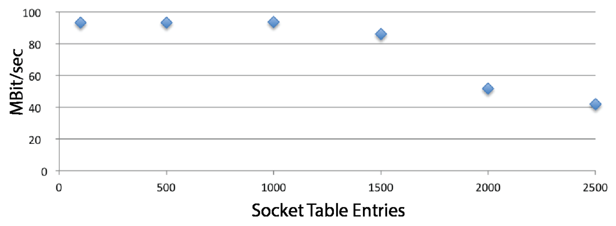

Fig. 9. Socket-Table Scalability

D. Lines of Code

Table II provides the lines-of-code (LOC) of each of the different mobility solutions. While LOC is not a conclusive metric in and of itself, it enables a quantitative comparison between mobility solutions in terms of relative complexity. The results highlight the simplicity of DIME, both in terms of total LOC and DIME's avoidance of kernel-level modifications.

Protocol	Kernel LOC	User LOC	Total LOC
MIPv6	XXX	XXX	XXX
HIP	0	28,770	28,770
DIME	200	2,400	2,600

TABLE II
LINES OF CODE

Other Features	MIPv6	HIP	DIME
IPv4 Support		✓	✓
UDP Support	✓	✓	✓
IPv4/v6 Handover		✓	✓
Private/Link Addrs			✓
Simultaneous Mob.	✓		✓
Preexisting Conns.	✓		✓
NAT Traversal		✓	*
Micro-mobility	✓		*
Multipath			✓
ARM Architecture			✓

∗with middlebox support

TABLE III
FEATURESET COMPARISON

E. Featureset Comparison

Though our evaluations thus far have focused on performance metrics, an equally important consideration is its qualitative featureset. We tested DIME, MIPv6 and HIP for support across a wide range of features, use-cases, and network environments that fall outside of standard mobility testbeds. Table III contains our results, and shows that while different proposals support different features, DIME is clearly the most adaptable and flexible proposal.

While some of these features (e.g., private IP addressing) may be simple implementation issues, others represent fundamental architectural limitations. Specifically, we highlight the inability of any other proposal to support ARM-based architectures (i.e., Raspberry Pis) as an important, and surprisingly fundamental, limitation of all other proposals. The high cost of porting a complex, deeply-integrated codebase prevents implementations of MIPv6 from making their way to the specialized system architectures used by resource-constrained devices, yet HIP's heavy reliance on cryptographic operations renders it completely unusable for such environments. In contrast, DIME avoids both problems and is completely out-of-the-box deployable on a stock Raspbian distribution.

V. CONCLUSIONS AND FUTURE WORK

We described DIME, the first solution for the seamless support of host mobility in the Internet. DIME is based on two main components, StackTrans in the data plane and IHMP in the control plane, which work together to create a lightweight, flexible, scale-free, and deployable approach to Internet host mobility.

StackTrans enables IP datagrams to be dynamically readdressed as a host moves throughout the Internet *without* requiring changes to applications, the DNS, transport-layer protocols, the network stack, the network layer, or intermediate nodes. StackTrans incurs minimal overhead, is deployable on a stock OS, and enables mobility signaling itself to be enacted out-of-band via IHMP (Internet Host Mobility Protocol) a simple signaling protocol. IHMP is far more lightweight than all prior solutions, scale-free with respect to the number of connections at a host, and explicitly does not depend on or require an additional host-identifier namespace or changes to an existing set of directory services (e.g., the DNS).

Our evaluations show that MIPv6 and HIP suffer from at least one vital weakness. HIP supports more application use-cases, but incurs severe performance penalties and requires a relatively static and brittle configuration at end hosts. MIPv6 is by far the least flexible solution and effectively cannot be deployed on today's systems. By contrast, DIME is seamlessly deployable on top of a wide range of systems, and outperforms the other solutions across a wide range of performance metrics.

References

[1] V. Cerf and R. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 1974.

[2] A. Lella. Number of Mobile-only Internet Users Now Exceeds Desktop-only in The U.S. https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S.

[3] R. Qureshi. Ericsson Mobility Report: On The Pulse of The Networked Society, 2014.

[4] C. Perkins and D.B. Johnson. Mobility Support in IPv6. *Proc. 2nd International Conference on Mobile Computing and Networking*, pages 27–37, 1996.

[5] R. Moskowitz et. al. Host Identity Protocol. *RFC 5201*, April 2008.

[6] W.M. Eddy. At What Layer Does Mobility Belong? *IEEE Communications Magazine*, 42(10):155–159, 2004.

[7] E. Perera, V. Sivaraman, and A. Seneviratne. Survey on Network Mobility Support. *Proc. ACM SIGMOBILE*, 2004.

[8] P. Bhagwat and C. Perkins. Network Layer Mobility: An Architecture and Survey. *Personal Communications*, 1996.

[9] D. Le, X. Fu, and D. Hogrefe. A Review of Mobility Support Paradigms for The Internet. *IEEE Communications Surveys & Tutorials*, 8(1):38–51, 2006.

[10] M. Komu, M. Sethi, and N. Beijar. A Survey of Identifier-Locator Split Addressing Architectures. *Computer Science Review*, 2015.

[11] D. Saha, A. Mukherjee, I.S. Misra, and M. Chakraborty. Mobility Support in IP: A Survey of Related Protocols. *IEEE Network*, 18(6):34–40, 2004.

[12] Z. Gao, A. Venkataramani, and J.F. Kurose. Towards a Quantitative Comparison of Location-Independent Network Architectures. In *ACM SIGCOMM Computer Communication Review*, 2014.

[13] V. Ishakian, I. Matta, and J. Akinwumi. On the Cost of Supporting Mobility and Multihoming. *Proc. GLOBECOM Workshops*, 2010.

[14] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. *IETF RFC 5533*, 2009.

[15] R. Atkinson, S. Bhatti, and S. Hailes. ILNP: Mobility, Multi-homing, Localized Addressing and Security through Naming. *Telecommunication Systems*, 42(3-4):273–291, 2009.

[16] C. Perkins. Mobile IP. *IEEE Communications Magazine*, 35(5):84–99, 1997.

[17] M. Kunishi, M. Ishiyama, K. Uehara, and H. Esaki. LIN6: A New Approach to Mobility Support in IPv6. *Proc. International Symposium on Wireless Personal Multimedia Communications*, 2000.

[18] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How Hard Can It Be? Designing and Implementing a Deployable multipath TCP. *Proc. USENIX NSDI*, pages 29–29, 2012.

[19] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Highly Available Internet Services Using Connection Migration. *Proc. International Conference on Distributed Computing Systems*, pages 17–26, 2002.

[20] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *ACM SIGCOMM Computer Communication Review*, pages 19–43, 1997.

[21] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. *Proc. International Conference on Distributed Computing Systems*, pages 136–143, 1995.

[22] D. Funato, K. Yasuda, and H. Tokuda. TCP-R: TCP Mobility Support for Continuous Operation. *Proc. International Conference on Network Protocols*, pages 229–236, 1997.

[23] S. Freire and A. Zúquete. A TCP-Layer Name Service for TCP Ports. *Proc. USENIX Annual Technical Conference*, pages 275–280, 2008.

[24] A.C. Snoeren and H. Balakrishnan. An End-to-End Approach to Host Mobility. *Proc. 6th International Conference on Mobile Computing and Networking*, pages 155–166, 2000.

[25] D.A. Maltz and P. Bhagwat. MSOCKS: An Architecture for Transport Layer Mobility. *Proc. IEEE INFOCOM*, 1998.

[26] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a Transport Protocol for Real-Time Applications. *RFC 3550*, July 2003.

[27] E. Wedlund and H. Schulzrinne. Mobility Support Ssing SIP. *Proc. ACM WoWMoM*, 1999.

[28] H. Schulzrinne and E. Wedlund. Application-Layer Mobility Using SIP. *Mobile Computing and Communications Review*, 4(3):47–57, 2000.

[29] A. Dutta, F. Vakil, J. Chen, M. Tauil, S. Baba, N. Nakajima, and H. Schulzrinne. Application Layer Mobility Management Scheme for Wireless Internet. *Proc. IEEE 3G Wireless*, 2001.

[30] S. Pack, K. Park, T. Kwon, and Y. Choi. SAMP: Scalable Application-Layer Mobility Protocol. *IEEE Communications Magazine*, 44(6):86–92, 2006.

[31] J. Kristiansson and P. Parnes. Application-Layer Mobility Support for Streaming Real-Time Media. *Proc. IEEE WCNC*, 2004.

[32] V. Zandy and B. Miller. Reliable Network Connections. *Proc. ACM MOBICOM*, 2002.

[33] B.Y.L. Kimura and H.C. Guardia. TIPS: Wrapping The Sockets API for Seamless IP Mobility. *Proc. ACM Symposium on Applied Computing*, 2008.

[34] T. Okoshi, M. Mochizuki, Y. Tobe, and H. Tokuda. MobileSocket: Toward Continuous Operation for Java Applications. *Proc. IEEE ICCCN*, 1999.

[35] H. Balakrishnan et. al. A Layered Naming Architecture for The Internet. *Proc. ACM SIGCOMM*, pages 343–352, 2004.

[36] A. Ghodsi et al. Intelligent Design Enables Architectural Evolution. *Proc. ACM HotNets*, page 3, 2011.

[37] I. Stoica et al. Internet Indirection Infrastructure. *Proc. ACM SIGCOMM*, 2002.

[38] B. Ford. Breaking Up The Transport Logjam. *Proc. ACM HotNets*, 2008.

[39] E. Nordstrom et al. Serval: An End-Host Stack for Service-Centric Networking. *Proc. USENIX NSDI*, 2012.

[40] D. et al. Han. XIA: Efficient Support for Evolvable Internetworking. *Proc. USENIX NSDI*, 2012.

[41] T. Koponen et al. Architecting for Innovation. *ACM SIGCOMM Computer Communication Review*, 41(3):24–36, 2011.

[42] J. Ubillos et al. Name-Based Sockets Architecture. *IETF Draft*, 2010.

[43] Named Data Networking (NDN). http://named-data.net.

[44] J.J. Garcia-Luna-Aceves, M. Mirzazad-Barijough, and E. Hemmati. Content-Centric Networking at Internet Scale through The Integration of Name Resolution and Routing. *Proc. ACM ICN '16*, 2016.

[45] S. Sevilla and J.J. Garcia-Luna-Aceves. Allowing applications to evolve with the internet: The case for internet resource descriptors. *Proc. IEEE ICC '14*, 2014.

[46] S. Sevilla and J.J. Garcia-Luna-Aceves. HIDRA: Hiding Mobility, Multiplexing, and Multi-homing from Internet Applications. *Proc. 17th IEEE Global Internet Symposium*, 2014.

[47] S. Sevilla and J.J. Garcia-Luna-Aceves. Freeing The IP Internet Architecture from Fixed IP Addresses. *Proc. IEEE ICNP '15*, 2015.

[48] T. Henderson, P. Nikander, and M. Komu. Using The Host Identity Protocol with Legacy Applications. *RFC 5203*, September 2008.

[49] J. Saltzer, D. Reed, and D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 1984.

[50] B. Ford. Directions in Internet Transport Evolution. *IETF Journal*, 2007.

[51] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no Longer Considered Harmful. *Proc. USENIX OSDI*, 2004.

[52] D. Phoomikiattisak and S. Bhatti. Mobility as A First Class Function. *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, 2015.

[53] IP Latency Statistics.
http://www.verizonenterprise.com/about/network/latency/.

[54] S. Sevilla and J.J. Garcia-Luna-Aceves. A Deployable Identifier-Locator Split Architecture. *Proc. IFIP Networking '17*, 2017.