A Deployable Identifier-Locator Split Architecture

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

***Abstract*—Despite the vast set of prior work on identifier-locator split architectures, no one approach has seen much success, adoption, or deployment in the Internet at large. We identify the key set of challenges that have inhibited the deployment of these proposals to date, and introduce the Dynamic Internet Mobility for End-Systems (DIME) approach. DIME is based on dynamic address translation between the transport and network layers of end hosts, combined with a simple out-of-band protocol that updates host-address bindings as needed. DIME is the first and only proposal that achieves a clean identifier-locator split without requiring modifications to the end-host OS or applications; modifications to existing network protocols, security mechanisms, or hardware; or a new host-identifier namespace. We evaluate a Linux daemon implementation of DIME, and show that it i outperforms existing mobility proposals such as mobile IP (MIPv6), multipath TCP (MPTCP), and the Host Identity Protocol (HIP) across a wide range of performance metrics.**

I. Introduction

Despite the rapid growth and proliferation of mobile devices, the Internet today still does not support seamless host mobility: connections must be restarted when a host changes network attachment points, and cannot take advantage of multiple network paths simultaneously. This inability stems primarily from the observation that higher-layer protocols (i.e., TCP and UDP) use IP addresses to consistently *identify* communicating processes in hosts, whereas 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. This separation is known as the *identifier-locator split*.

The challenge of splitting identifiers from locators has attracted a large amount of prior work. However, despite the well understood architectural benefits, no one approach to the identifier-locator split has seen widespread adoption or deployment in the Internet at large. Section II summarizes the prior work in this area with an eye to this deployment gap, and identifies the primary requirements that such a solution must exhibit.

Sections III to V introduce the *Dynamic Internet Mobility for End-Systems* (**DIME**) approach based on the requirements identified in Section II. DIME is based on the key insight that the addresses used by the transport and application layers need not be the same as the addresses used in the data plane, and is the first approach to Internet host mobility that is completely seamless with respect to applications, the Internet routing infrastructure, and the control and data planes of existing protocols used in the Internet. As such, DIME is the only solution that 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.

ISBN 978-3-901882-94-4

DIME consists of two key architectural components. First, *StackTrans* is a modified form of network address translation (NAT) between the transport and network layers of the stack, such that the transport layers and above see unmodified network addresses (i.e., identifiers), yet datagrams are addressed to the current network address of a host (i.e., locators). Second, the *Internet Host Mobility Protocol* (IHMP) is an out-of-band, end-to-end signaling protocol that updates the address bindings used by StackTrans as hosts gain and lose network addresses.

Section VI describes a concrete Linux implementation of DIME and presents detailed comparisons with Mobile IP [1], Multipath TCP [2], and the Host Identity Protocol [3]. Our results show that (1) DIME is much more lightweight and deployable, (2) DIME runs on a wider range of hardware and software, and (3) DIME is significantly more scalable across a wide range of metrics.

II. Related Work and Design Requirements

All prior approaches to Internet host mobility can be categorized based on how the identifier-locator split is handled in the data plane. We group prior work into approaches that transmit host identifiers, approaches that transmit host locators, and approaches that transmit both identifiers *and* locators.

A. Host-Identifier Approaches

The first proposals for Internet host mobility preserved the identifier bindings made by higher layers, and approached host mobility entirely within the network itself - either by updating a node's location in routing tables as it moves (e.g., Mobile IP [1]) or by performing a modified form of NAT in the network core (e.g., LISP [4]).

The key benefit of these host-identifier approaches is that by restricting all modifications to the network layer, they are able to seamlessly maintain the bindings, connections, and APIs of higher-layer protocols (i.e., TCP and application-layer protocols) at end hosts without modification. However, these approaches also create significant problems, including (1) untenable control signaling and routing-table growth via indirection; and (2) increased network bandwidth consumption and latency via triangle-routing.

Most importantly, these approaches *also* require replacing existing network infrastructure (i.e., routers) because they alter network-layer protocols in both the control *and* data planes. Such an overhaul is not feasible, and leads us to the following first requirement.

R1: Unmodified Routing Infrastructure: *A deployable identifier-locator split must run over* all *existing routers and switches without requiring their modification or replacement.*

B. Host-Locator Approaches

In response to the inherent problems with host-identifier proposals, recent host-locator proposals [5], [2], [6], [7], [8] split identifiers from locators *above* the network layer at end hosts. These approaches propose addressing datagrams to the current network location of a host (i.e., the locator), and updating the IP address bindings made by higher layers (typically at the transport layer) of end hosts by way of additional signaling or protocol options exercised when a host experiences mobility.

Host-locator approaches have seen slightly more success than host-identifier approaches (e.g., MPTCP is actively being developed on both Apple and Linux) primarily because they do not require changes to the routing infrastructure, and hence can be deployed entirely at end hosts in software, as opposed to hardware. However, these approaches are almost all TCP-specific and are not backwards compatible with unmodified applications, operating systems (OS), or NAT boxes. Additionally, these approaches require applications to be rewritten *and* significant modifications to kernel code in the network stack, which in turn requires an untenable amount of independent development effort and support *before* any benefits can be realized.

This non-trivial amount of developer overhead, combined with the lack of a compelling "killer app," results in a chicken-and-egg situation that discourages the development of such proposals and prohibits an incremental "opt-in" deployment path. The failure of these host-locator approaches leads us to our second requirement.

R2: Unmodified OS and Applications. *Our solution* must *be incrementally deployable and must* not *require buy-in from or modifications to applications or the host operating system.*

C. Combined Approaches

Many recent approaches to Internet host mobility transmit identifiers *and* locators in the data plane, with the intent that higher layers only use identifiers and the network layer only uses locators. This is achieved by defining a new host-identity layer and including this identity layer in one of three ways: (a) as a part of the IPv6 address space (e.g., ILNP [9]), (b) between the network and transport layers (e.g., HIP [3] and many Future Internet proposals [10], [11], [12], [13]) or (c) in the application layer (e.g., SIP [14] or others [15], [16]).

Critically, all of these proposals require the convergence on, and standardization of, a new host-identifier namespace to be inserted into the network stack.[1] This represents a massive adoption hurdle, and would essentially require the redesign and replacement of network applications, operating systems, and middleboxes. Second, if adopted, any of these proposals would "lock in" a new set of endpoint identifiers as part of the new Internet architecture, which raises significant concerns about the tenability or evolvability of said approaches.

The improbable adoption of a new host-identifier name space, combined with the associated concerns regarding identifier lock-in, leads us to our third requirement.

R3: No New Namespace. *For an identifier-locator split to evolve organically and incrementally, it must not depend on the development and standardization of a new identifier namespace.*

[1]While many works propose leveraging the DNS as an existing namespace; [17] makes a strong argument for why the DNS is fundamentally unable to support host address mobility.

III. StackTrans and the Data Plane

While many of the proposals in Section II satisfy one or two of the stated requirements for seamless support of Internet host mobility, no proposal to date satisfies all three. To resolve this challenge, we argue that what is needed is a "drop-in" solution that can be incrementally deployed by individual users or network operators as necessary. It follows that a satisfactory solution must: (a) be fully contained at end hosts, because we cannot rely on changes to network infrastructure; (b) seamlessly support unmodified applications and be deployable on top of a stock OS; and (c) be solely based on existing identifiers (i.e., IPv4 or IPv6 addresses). DIME is designed specifically to satisfy these three characteristics.

Figure 1 illustrates the main architecture of DIME, which is comprised of two separate components that work together: a data-plane readdressing technique called *StackTrans*, and a lightweight out-of-band signaling protocol called the Internet Host Mobility Protocol (IHMP). This section discusses StackTrans, and Section IV discusses IHMP.

The goal of StackTrans is to dynamically readdress datagrams as they pass between the transport and network layers of the stack. This way 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 a two-step address translation process, illustrated in Figure 2, that first maps the IP address bound by a socket to an entity called a *Host Identifier* (HID), and then maps the HID to its current network address.

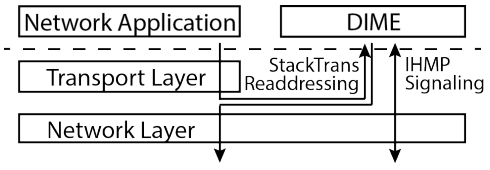

Fig. 1. DIME System Architecture

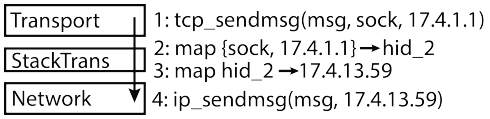

Fig. 2. StackTrans Packet Readdressing

(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

A. The Socket Table

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 or receives data from a foreign address `faddr` the address is looked up in the HID Table and mapped to an existing HID. If no such HID exists, a new one is created. Once the HID is obtained or generated, a {socket, faddr, hid} tuple is stored in the socket table, and used to ensure that all communication from the socket to `faddr` is mapped to the correct HID, and all communication from the HID to the socket is mapped back to `faddr`.

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. For example, socket s opens a connection to address $a1$, the foreign host moves from $a1$ to $a2$, and then socket t opens a connection to address $a2$.

Multiplexing {socket, faddr} tuples to a HID, rather than directly to an address, achieves two key benefits. First, it enables multiple addresses for a host to be stored and managed in a unified, centralized, protocol-independent way. Second, given that multiple {socket, faddr} tuples can bind the same HID, any updates to the address-set of a HID immediately affect all connections to that host without additional signaling.

B. The Host Identifier Table

A Host Identifier is an internally-kept value that semantically refers to a foreign host. While the architectural concept of a host identifier is not new, all prior works have implemented such identifiers as a new global namespace and layer in the network stack. As discussed above, this violates R3 and represents a significant deployment roadblock. In contrast to these implementations, the Host Identifiers used by DIME are simply internal indices into the HID Table: HIDs have no external value, are not operated on or seen by higher layers, and are never propagated over the network.

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 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. Finally, the HID of a host is also optionally bound to a *host key* for the host. The reason for storing active addresses is obvious (i.e., data-plane communication), and the other address sets are used mainly for the IHMP signaling protocol described in Section IV.

C. Data-Plane Security

Given that StackTrans modifies datagrams between the transport and network layers at end hosts, it is completely orthogonal to all existing approaches to data-plane security. Such lower-layer security protocols as IPSec encrypt the IP datagram *below* the translation, and are therefore oblivious to the presence of StackTrans. On the other hand, such higher-layer security protocols as TLS encrypt the datagram *above* the translation, using the initially-bound identifiers that are then recreated at the other end of the connection before any security checks are made. This contrasts with the mobility protocols in Section II, which typically include end-to-end security as an integral component of the protocol itself.

While data-plane security is important, we argue that it is equally important to separate it from network mobility. The StackTrans approach enables both types of security protocols to evolve independently of StackTrans, and *also* allows mobility to be deployed without requiring specific security solutions. This split is important for many different network scenarios, including (1) resource-constrained devices that cannot support robust cryptographic operations and (2) physically isolated or secured networks where data-plane security is unnecessary. Finally, this orthogonal approach enables the control plane to be secured separately from the data plane; we discuss this design further in Section IV-F.

IV. The Internet Host Mobility Protocol

StackTrans enables connections to be dynamically redirected at end hosts without requiring modifications to the network stack or data plane. In turn, this allows host mobility signaling to be taken out of the data plane and enacted as a simple end-to-end signaling protocol, which we call IHMP. Just like a routing protocol runs out of band with respect to the forwarding of datagrams, IHMP runs out of band with respect to the end-to-end protocols in the data plane.

The IHMP daemon 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 IHMP messages encapsulated in UDP. The choice of running IHMP on top of UDP was threefold. It keeps IHMP transactions lightweight, allows IHMP to interpret dropped messages as a sign that a path may not be sufficiently reliable, and in contrast to ICMP enables NAT detection and traversal.

When a host receives an IHMP message from a foreign host, it updates its HID table to reflect the changes stated in the message, at which point StackTrans immediately incorporates them into the data path.

Table I lists the different IHMP message types, and Figure 4 illustrates the IHMP message format. The control field corresponds to the message type in Table I, and the sequence

Control Code	Message Type
0	HELLO
1	PATH_PROBE
2	ADDR_UP
3	ADDR_UP_UNREACHABLE
4	ADDR_DOWN
5	ADDR_DOWN_UNREACHABLE
6	HANDOFF
7	HANDOFF_UNREACHABLE
8	ACK
9	ROUTER_ACK

TABLE I
IHMP MESSAGES

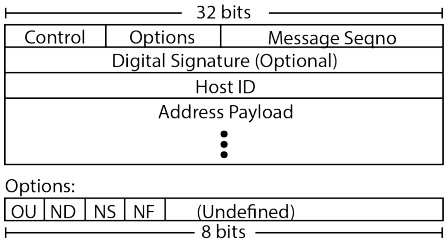

Fig. 4. IHMP Message Format

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.

A. End-To-End Host Identification

The Host ID field in IHMP is a major departure from all other proposals that use end-to-end updates. Whereas all prior proposals rely on a separate host identifier namespace (e.g., a DNS hostname or a host-identity tag) to consistently identify a host across network address changes, IHMP is specifically designed *not* to assume the existence of any such namespace.

Not requiring a specific namespace for hosts makes IHMP much more lightweight and deployable across a wider range of networks. However, this also means that the *only* way for a receiving host to identify a sender is by the IP address stored in the Host ID field. The sender populates this field with an IP address chosen from the set of *Local Addresses* bound to the HID of the receiver. This process ensures that the IP address will multiplex to the correct HID at the receiver, and is the reason for storing the local addresses reachable by a foreign host. It is critical to point out that this solution does *not* require globally-unique IP addresses! Rather, it only requires that an IP address uniquely refer to the sender *from the perspective of the receiver* - fortunately, this is a valid assumption that underpins all network-layer protocols and routing.

B. Hello Exchange

When an active host creates a new HID table entry, it must: (a) query the foreign host for the presence of other network addresses, and (b) advertise its set of local addresses to the foreign host. The active host accomplishes this via a simple two-way HELLO message exchange, which is 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 a foreign host receives a HELLO message, it creates an entry in its HID table for the active host that sent the message, adds the source address of the message to the set of active addresses of the HID, 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. Next, it sends an ACK back to the active host at the same address used to send the HELLO. The ACK also contains all other addresses the foreign 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, and all other advertised addresses to the set of unreachable addresses.

In addition to probing for other network addresses, the HELLO exchange also tests for IHMP support. If no ACK is received after a timeout, the active host retransmits the HELLO message two more times before concluding that the foreign host does not support IHMP, at which case the data-plane communication falls back to the standard (non-translated) path.

C. Back-path Probing

From the perspective of a foreign host, the other addresses included in a HELLO message are either: (a) guaranteed reachable (e.g., a publicly reachable IP address), (b) guaranteed unreachable (e.g., in a network that the host cannot reach), or (c) *potentially* reachable (e.g., a private network that the host also has an address in). The foreign host sorts the addresses in cases (a) and (b) into active and unreachable addresses, respectively, and manages the addresses in case (c) by sending a PATH_PROBE message as illustrated in Step 3 of Figure 5.

In our example, the active host initially reaches the foreign host over the public Internet, and indicates in its HELLO message that it also has addresses in networks 192.168.0.0/16 and 10.0.0.0/8. Subsequently, the foreign host sends PATH_PROBE messages over the 192.168.0.0/16 and 10.0.0.0/8 networks to which it is connected. When the active host receives a PATH_PROBE message, it updates the set of active and local addresses for the HID to reflect reachability, and replies along the same path with an ACK (Step 4 of Figure 5). The foreign host does the same upon receipt of the ACK.

D. Address-Up and Address-Down Events

When a host gains a network address, it sends an ADDR_UP message from this address to the foreign host; this message explicitly encodes the new address in the IHMP message.[2] When the foreign host receives and ADDR_UP message, it adapts its HID table and responds with an ACK.

If the active host does not receive an ACK in an acceptable amount of time, it determines that the foreign host is *not* reachable from the new network address. In this case, the active host sends an ADDR_UP_UNREACHABLE message with the new address to the foreign host on any available network

[2]This explicit address encoding allows us to detect and mitigate NATs; we discuss this process further in Section V-D.

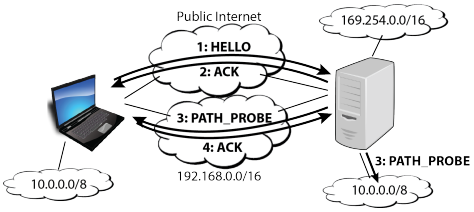

Fig. 5. HELLO message exchange

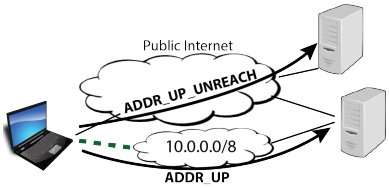

Fig. 6. Address-up signaling

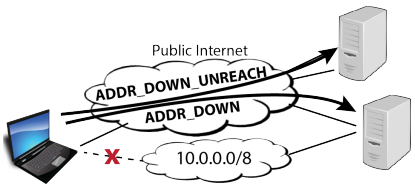

Fig. 7. Address-down signaling

address. The intent of this address is to ensure that the HID table of the foreign host is accurately updated; hence, the foreign host must respond to such a message with an ACK.

When a host *loses* a network address, it checks the local address set for each HID to determine if it is still reachable, and sends an ADDR_DOWN or ADDR_DOWN_UNREACHABLE message as appropriate. In both cases, the foreign host must respond with an ACK. If a lost address effectively disconnects the active host from the foreign host, it can wait for an amount of time before removing the HID entry and reporting an error to network applications.

E. Handoffs

Handoff events effectively combine ADDR_UP and ADDR_DOWN events into one. The network addresses gained and lost 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. As with address-up events, the active host sends a HANDOFF message from the new network address to probe reachability, and encodes both the new and old network addresses in the payload. In the cases where the foreign host is not reachable by the new network address, a HANDOFF_UNREACHABLE message is sent in its place. For clarity and consistency, handoff messages use an OU (OLDADDR_UNREACHABLE) flag in the options filed to indicate whether the *old* address was reachable by the passive host or not.

F. Control-Plane Security

Spoofed IHMP messages have the potential to disrupt and redirect communication. To address this, IHMP uses optional digital signatures to ensure message *integrity* without *confidentiality*. This decision reduces computational overhead and allows intermediate nodes (i.e., middleboxes) to read the content of IHMP messages, and generate control messages signed with their own key without violating the security model. We discuss these optimizations in Section V-C.

The IHMP daemon signs outgoing messages with its private key, and can validate inbound messages by binding the correct public key to a HID. There exist several different methods for obtaining the public key for a host, just as there exist several different security policies regarding which keys and messages a host will accept. However, such topics are outside the scope of this paper, except to mention that IHMP is compatible with all public-key solutions, and IHMP explicitly does not *depend* on such a solution and is deployable even in network scenarios that do not require or support security protocols.

V. Additional Considerations and Edge-Cases

A. Simultaneous Mobility

The lack of rendezvous nodes or integration with a name-resolution service in DIME poses a problem for *simultaneous mobility* cases in which both nodes change network addresses before the end-to-end signaling of IHMP can converge. We note that this case is remarkably uncommon in the client-server communication paradigm that underpins the vast majority of network communication, but still merits attention. Depending on the severity of the alteration to the address set, we describe communication between the hosts as either *partially* disrupted or *fully* disrupted, depending on whether at least one of the hosts has kept at least one of its network addresses.

DIME automatically recovers from partial disruption and converges without difficulty. Upon not receiving an ACK, the fully-mobile host simply re-sends the same mobility message to the other addresses of the host, until it receives an ACK from the address that was maintained. The partially-mobile host then transmits its other network addresses in a subsequent exchange. DIME is unable to recover from *fully* disrupted simultaneous mobility, wherein neither host maintains any addresses, because both address sets are completely incorrect; in this case, the client must discover the new IP address of the server through some other service.

B. Mistaken Identities

Given that DIME does not rely on a unique or separate host namespace, it is vulnerable to the following mistaken identity scenario: (1) Host A is using address $a1$ to communicate with host X; (2) A moves from $a1$ to $a2$ but is unable to tell X; (3) host B obtains the address $a1$ of A, and (4) B sends a message from $a1$ to host X.

DIME guards against mistaken-identity scenarios by verifying that the sequence number is correct before processing the update, and by sending an ACK with the IS (INCORRECT_SEQNO) bit set if this is not the case. This supports end-to-end recovery of dropped packets, but also allows the non-mobile host (X in the prior example) to create a new HID entry in the event that a host ungracefully departed. Note that this approach guards only against accidental scenarios and is clearly vulnerable to replay attacks. Hence, if the network environment is considered insecure or hostile, digital signatures should be used.

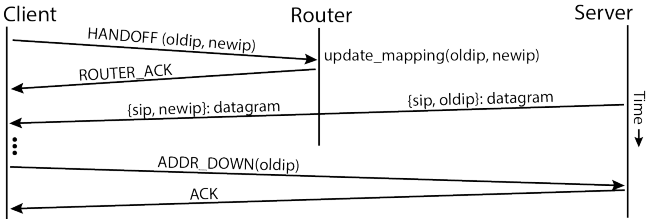

Fig. 8. Micro-Mobility Signaling

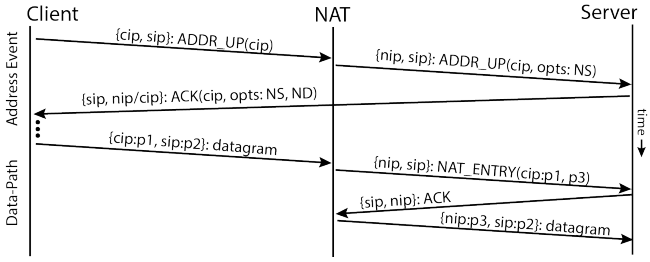

Fig. 9. Mobility signaling with NAT

C. Micro-Mobility

While DIME is an end *host* solution, it is not necessarily end-to-end. DIME-aware network entities such as middleboxes and routers are able to intercept IHMP messages that add a new address (either an `ADDR_UP` or `HANDOFF` message) and respond with a `ROUTER_ACK` message. Depending on topology or policy, the middlebox can either enact micro-mobility by updating or installing a new routing-table entry, as in Figure 8, or indicate that the update is to be rejected.

Designating a separate `ROUTER_ACK` message type acknowledges middleboxes and routers as first-class citizens of the Internet, and provides them with an architectural location to integrate with DIME. This enables DIME to support micro-mobility without any end-to-end signaling, and supports proper security policy by allowing the `ROUTER_ACK` message to be signed with a separate key. Finally, by explicitly informing the mobile host that its update was processed by an intermediate router and *not* the end-host, the mobile host knows that the HID table of the foreign host still contains the old address. This is illustrated at the bottom of Figure 8, where the host loses `newip` but sends an `ADDR_DOWN` message containing `oldip`.

D. NAT Detection and Traversal

While `ROUTER_ACK` messages support mobility *within* subnetworks and behind NATs, they are insufficient to support mobility into or out of NATs, specifically because NATs enact a many-to-one address mapping through transport-layer port renumbering. Given that NATs are so widespread in the Internet today and expected to be "here to stay" for the foreseeable future [18], DIME provides a mechanism for NAT detection and traversal in all cases, even those in which the NAT is DIME-unaware.

DIME detects NATs by explicitly encoding the source address in the DIME message body itself, and stores NAT addresses in the HID table as a `nat_addr:host_addr` tuple; this enables correct end-to-end signaling as hosts move in and out of NATs. NATs that support DIME indicate this by setting a `NS` (`NAT_SUPPORTED`) flag in all IHMP messages that traverse the NAT; these NATs send a `NAT_ENTRY` message to the public host as they create new port-mappings for each connection, as illustrated in Figure 9. This mapping is stored at the end-host and used to map the NATed port back to the original source port for delivery.

DIME handles legacy NATs that do not support DIME by setting a `ND` (`NAT_DETECTED`) flag in the `ACK`. Since DIME cannot migrate connections into DIME-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 marked as such, and no further signaling occurs (i.e., communication falls-back to normal operation without StackTrans).[3]

VI. Implementation and Evaluation

We implemented DIME as a user-space daemon and Loadable Kernel Module (LKM), which satisfies R2 because our LKM is fully installable on a stock OS. We deployed our code on two laptops running Ubuntu, the *Mobile Host* (MH) and the *Corresponding Host* (CH), and connected them with a switch (s_1) and two routers (r_1 and r_2) to create the topology shown in Figure 10. 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 [19], [20].

To evaluate and compare performance, we conducted a standard mobility experiment in which MH moves from r_1 to r_2 while conducting a throughput test at CH. To provide more consistent results and remove variance, we induced address-up and address-down events programmatically via the `ip` command, with a five-second gap between each event. We conducted each experiment ten times and provide mean values. We compared DIME against three protocols, Mobile IP (MIPv6), Multipath TCP (MPTCP), and the Host Identity Protocol (HIP), which we specifically chose as "flagship" examples to represent each of the three categories listed in Section II. We recognize that MPTCP does not specifically seek to separate identifiers from locators for the sake of mobility; however, it provides an architecturally similar design and has a well-developed and accessible codebase.

A. Deployment and Configuration

Our initial and most striking finding was that the different protocols varied wildly in terms of how much effort it took to configure even the basic testbed in Figure 10. Table II provides a rough summary of these protocols, and shows that MIPv6 stands out by far as the most fragile and ossified approach. The

[3]We acknowledge that there may exist other techniques to mitigate the identifier-locator split across NATs. For brevity and focus, we omit further discussion of this topic, and leave it as a topic for future work.

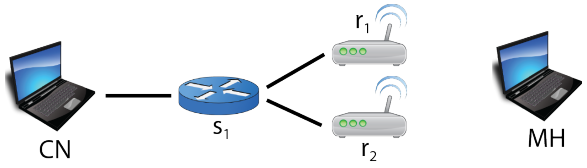

Fig. 10. Testbed topology

Requirement	MIPv6	MPTCP	HIP	DIME
Daemons	4	0	1	1
Config. Files	3	1	2	1
App Mods.			✓	
System Configs	✓	✓		
Custom Kernel	✓	✓		
Router Mods.	✓			

TABLE II
DEPLOYMENT REQUIREMENTS

reliance of MIPv6 on deep kernel integration requires a custom kernel and a user-space daemon at the end hosts. However, the 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 r_1 and r_2, as well as the end hosts.

Installing and configuring MPTCP and HIP were both much easier, requiring a custom kernel or user-space daemon, respectively, at the end hosts. However, in practice, we still found concerning implementation issues. MPTCP required us to download, compile, and install a custom Linux kernel, which is not an easy task for the average user, and it's reliance on link selection by source address binding forces an unorthodox configuration wherein separate routing tables are maintained for each interface. While this did not impact our simple testbed, it raises the question of the ability of MPTCP to support more unorthodox, dynamic, or virtualized network configurations. HIP configuration relies heavily on manually encoding static, preconfigured bindings at both end hosts,[4] and the use of the 1.0.0.0/8 block for LSI bindings in HIP raises 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).

Finally, we found that DIME "simply works." DIME exists as a single, standalone user-space daemon, requiring no configurations or modifications to application binaries or the underlying OS. The leverage of existing IP address bindings in DIME enables it to work without the need for specific namespace bindings or "pseudo" IP addresses, and its 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. Handoff Latency

We measured the latency needed to complete a handoff using two metrics: (a) the time it takes the protocol to identify and respond to the network event (i.e., the time elapsed from the network event to the first transmitted control message), which we call *Control Latency*; and (b) the time it takes the protocol to successfully handoff the connection (i.e., the duration from the first control message to the first correctly addressed datagram), which we call *Dataplane Latency*.

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

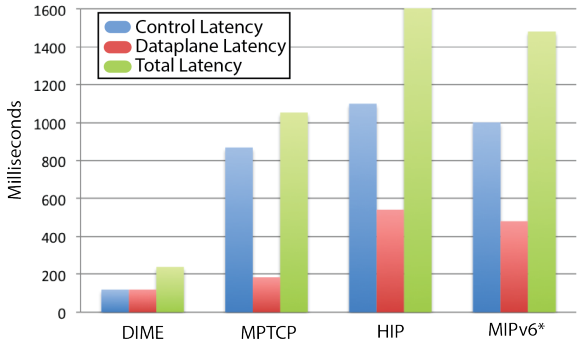

Fig. 11. Handoff Signaling Latency

The results in Figure 11 show that in all types of latency, DIME dramatically outperforms all other approaches. In terms of control latency, we found that both MPTCP and HIP wait approximately one second to start the mobility handshake process. We attribute this delay to the reliance on message retransmission timeouts and the need to be conservative with mobility signaling. In contrast, DIME is implemented as a user-space daemon, has direct integration with network-layer events, and supports graceful failure. This allows DIME to be much more aggressive with its network signaling and probing.

Dataplane latency is calculated from the moment the first message is sent over the network (i.e., the end of control latency) to the first correctly-addressed datagram; hence, it is not influenced by control latency. We found that our results for dataplane latency in Figure 11 very closely tracked the expected RTTs needed for the mobility signaling protocol to complete, and highlight the simpler handshake of DIME, as well as the lack of cryptographic operations needed to migrate the connection.

The asterisk in Figure 11 indicates that the control latency of MIPv6 varied dramatically, because the mobility exchange is triggered by the router (via RA messages) and not the host. In the interest of providing a competitive comparison, we provided results collected on the minimum possible RA interval of one second, but stress that these conditions are overly optimistic, too chatty, and unlikely to be used in any actual deployment of MIPv6.

C. Connection-Establishment Latency

We examined the additional latency (if any) required to *establish* a mobile connection versus a traditional one. Interestingly, we found that in all cases except HIP, the additional latency was minimal (<4ms), whereas HIP incurred dramatic connection establishment latency (400+ms for UDP and 800+ms for TCP). We attribute this to the fact that HIP uses a four-way-handshake to establish a HIP communication session *independently* of the three-way-handshake of TCP,

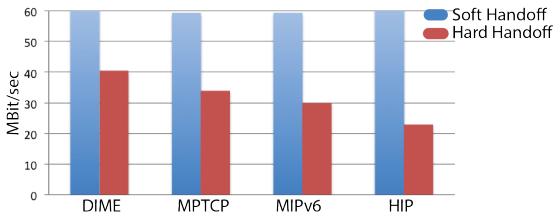

Fig. 12. TCP Handoff Goodput

Protocol	Messages Sent
IHMP	2 * max($nAddrs_{local}$, $nAddrs_{host}$)
MPTCP	4 * $nAddrs_{local}$ * $nAddrs_{host}$ * $nConns$
HIP	4 * $nAddrs_{local}$ * $nAddrs_{host}$
MIPv6	[2 (to HA) + 6 (to CH)] * $nAddrs_{local}$

TABLE III
HANDOFF CONTROL MESSAGES

whereas other approaches either conduct mobility signaling outside of the data plane (MIP and DIME) or integrate it with the existing TCP handshake procedure (MPTCP).

D. Data Plane Throughput

Figure 12 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 in the interest of comparing against MPTCP. We note that UDP goodput results collected for DIME, MIPv6, and HIP were all similar to the TCP results shown.

Figure 12 shows that each protocol provides almost identical results over a soft handoff. This is because each protocol has enough time to compete the handover signaling and maintain a constant data-rate (bound by the physical links) when the first address is lost, despite the different architectures or handoff signaling used.

It is intuitive that the maximum possible goodput for the hard handoff scenario will be approximately 2/3 of the soft handoff, given that the disconnection lasts five seconds and the total transfer lasts fifteen. Figure 12 shows that DIME achieves this value almost exactly, and noticeably outperforms all other proposals. We attribute this result to a combination of the faster control message exchange and transparency with respect to TCP attained in DIME. In contrast, we found that the performance of MPTCP suffered from the additional latency studied in Section VI-B, as well as the slow-start algorithm. MIPv6 mitigates this slow-start by migrating the existing TCP connection without triggering congestion-control backoff, yet it suffers from a noticeably longer handoff procedure. Finally, we found that the large decrease in goodput with HIP appeared to be the result of "buffer bloat" at the HIP daemon during the disconnection, which in turn created erratic and unfavorable interactions with the congestion control algorithms of TCP for the remainder of the connection.

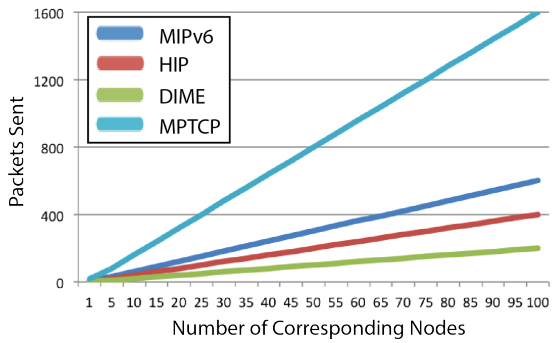

Fig. 13. Handoff Control Message Scalability

E. Control Message Scalability

We analyzed the number of control messages sent in response to a network handoff (soft or hard) and the factors that affect this number. Table III presents a formula for each protocol. The results show that IHMP outperforms its competitors, but *also* reveals three key aspects of how IHMP compares to the other approaches. First, the simplicity of the IHMP exchange results in a two-message handshake, as opposed to the four-message exchange used by HIP and MPTCP or the eight-message exchange of MIPv6 with RO. Second, the $nAddrs_{host}$ factor illustrates that both MPTCP and HIP do *address-pairwise* exchanges, wherein if hosts A and B both have two addresses, four exchanges are attempted: A1-B1, A1-B2, A2-B1, and A2-B2. In contrast, since IHMP stores addresses for a host and identifies reachability via routing-table information, it is able to accomplish the same example with two exchanges: A1-B1 and A2-B2.

The presence of $nConns$ in the MPTCP formula (which represents the number of active connections between the two hosts) highlights a fundamental drawback of supporting host mobility at the transport layer: Because the state of each connection is kept independently, the same procedure must be undertaken separately for each individual connection. This is an important consideration, especially because the average number of connections that a host has at any given point in time significantly outnumbers the number of separate hosts with which it is communicating, at a ratio of *at least* 4 to 1! [8] This has profound implications for the scalability of transport-layer solutions.

Figure 13 illustrates control message growth for a single handoff as we vary the number of corresponding hosts from 1 to 100. To be least favorable to DIME, the graph assumes that both hosts just have a single address, and makes a conservative estimate of $nConns = 4$. Even with these assumptions, we find that DIME significantly outperforms all existing approaches, and that MPTCP in particular incurs much more control signaling than the other approaches.

F. Requirements and Feature-Set Comparison

Though our evaluations thus far have focused on performance metrics, an equally important consideration is its qualitative feature set. In addition to evaluating the requirements enumerated in Section II, we extensively tested DIME and

Requirement	MIPv6	MPTCP	HIP	DIME
Unmodified Kernel			✓	✓
Unmodified Apps	✓	✓		✓
Unmodified Infra.		✓	✓	✓
Existing Namespaces	✓	✓		✓

Other Features	MIPv6	MPTCP	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 IV
FEATURESET COMPARISON

the other approaches for support across a wide range of features, use-cases, and network environments that fall outside of standard mobility testbeds. Table V 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 (e.g., adding UDP support to MPTCP) represent fundamental architectural limitations. Specifically, we highlight the inability of any proposal other than DIME 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 or MPTCP from making their way to the specialized system architectures used by resource-constrained devices. 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.

VII. CONCLUSIONS AND FUTURE WORK

We introduced DIME, the first seamlessly deployable identifier-locator split architecture. DIME is based on two main components, StackTrans in the data plane and IHMP in the control plane, which work together to enable a lightweight, flexible, and deployable approach to Internet host mobility.

StackTrans represents a dramatic departure from previous proposals, in that it enables datagrams to be dynamically readdressed as a host moves around the Internet *without* requiring changes to applications, 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 a simple signaling protocol, IHMP. IHMP is more lightweight by far than existing solutions, scale-free with respect to the number of connections at a host, and explicitly does not require an additional host-identifier namespace.

Our evaluation of prior solutions to Internet host mobility shows that they all suffer from at least one vital weakness. For example, MPTCP is clearly the current state-of-the-art for features such as NAT traversal and multipath routing, yet it cannot support UDP and scales poorly. HIP supports more application use-cases, but incurs severe performance penalties and requires a relatively static and brittle configuration at end hosts. Finally, MIPv6 is by far the least flexible solution and effectively cannot be deployed on today's systems. DIME addresses all these points, and is seamlessly deployable on top of a wide range of systems. DIME outperforms all other solutions across a wide range of performance metrics, including latency, throughput, and scalability. Equally important, DIME supports a much more robust feature-set.

Building on these experimental results, the true strength of DIME lies in its fresh approach to the old and fundamental problem of the identifier-locator split. DIME paves the way for a rich set of future work in many fields involving the Internet and protocol stacks in general, including: adapting existing identity-layer protocols such as HIP to use a StackTrans approach over encapsulation; adapting transport-layer protocols such as MPTCP to use IHMP to reduce and optimize control message signaling; and further integrating StackTrans to support features such as multipath routing.

REFERENCES

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

[2] 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.

[3] R. Moskowitz et. al. Host identity protocol. *RFC 5201*, April 2008.

[4] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). *IETF Experimental RFC 6830*, 2013.

[5] W.M. Eddy. At what layer does mobility belong? *IEEE Communications Magazine*, 42(10):155–159, 2004.

[6] 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.

[7] D.A. Maltz and P. Bhagwat. MSOCKS: An architecture for transport layer mobility. *Proc. IEEE INFOCOM*, 1998.

[8] S. Sevilla and J.J. Garcia-Luna-Aceves. freeing the IP internet architecture from fixed IP addresses. *Proc. IEEE International Conference on Network Protocols*, 2015.

[9] R. Atkinson, S. Bhatti, and S. Hailes. ILNP: mobility, multi-homing, localised addressing and security through naming. *Telecommunication Systems*, 42(3-4):273–291, 2009.

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

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

[12] E. Nordstrom et al. Serval: An end-host stack for service-centric networking. *Proc. USENIX NSDI*, 2012.

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

[14] E. Wedlund and H. Schulzrinne. Mobility support using SIP. *Proc. ACM WoWMoM*, 1999.

[15] J. Ubillos et al. Name-based sockets architecture. *IETF Draft*, 2010.

[16] V. Zandy and B. Miller. Reliable network connections. *Proc. ACM MOBICOM*, 2002.

[17] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav. A global name service for a highly mobile internetwork. *Proc. ACM SIGCOMM*, 2014.

[18] B. Ford. Directions in Internet transport evolution. *IETF Journal*, 2007.

[19] D. Phoomikiattisak and S. Bhatti. Mobility as a first class function. *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, 2015.

[20] IP latency statistics. http://www.verizonenterprise.com/about/network/latency/.