UNIVERSITY OF CALIFORNIA

SANTA CRUZ

IMPROVING THE INTERNET ARCHITECTURE THROUGH INDIRECTION AND VIRTUALIZATION

A thesis submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Spencer Sevilla

June 2017

The Thesis of Spencer Sevilla
is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Carlos Maltzahn

Professor Katia Obraczka

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Improving The Internet Architecture through Indirection and Virtualization

by

Spencer Sevilla

The current Internet architecture requires applications to transparently bind and manage network addresses and ports. This design creates and exacerbates several problems for the current Internet as well as its future evolution. These problems and challenges include (1) network address mobility and multihoming; (2) future Internet evolution; (3) service- or content-centricity; and (4) adding, subtracting, or evolving the layers in the network stack.

To address all of these problems, this thesis introduces a novel approach to Internet naming and addressing, which we call hidden identifiers. Hidden identifiers enable applications to semantically express the exact network resource they desire, and allows the operating system to subsequently bind and manage all other network concerns. In this manner, we provide integrated support for all the problem-cases enumerated above while simultaneously simplifying the network API presented to applications.

We introduce, implement, and evaluate HIDRA, the first network stack architecture based on hidden identifiers. We show that this network stack provides integrated support for features such as network mobility and multihoming, and explain how the HIDRA architecture can support a wide range of semantic bindings. Finally, we show how this feature can be leveraged to support the goals of information-centric networking on top of the existing TCP/IP network stack by slightly modifying DNS and HTTP.

Dedication

This thesis is dedicated to my late grandfather, Leslie Arthur Welge. He once told me that graduate school was where you learn more and more about less and less, and he'd certainly be thrilled to hear that I now know everything about nothing at all.

Acknowledgements

Much gratitude and acknowledgement goes to Sam. Regardless of what lab, office, or city we're in, you've been a fantastic coworker, labmate, and friend. I've always been grateful for your insight in networks, protocols, and systems, as well as the business end of things, and I sincerely look forward optimistically to collaborating with you in the future.

To the Buena Vista house, both old and new - Ashley, Clio, Kat, Colin, Kate, Sara, Ben, Kevin, Mike, and Stacey: anyone would be envious of such housemates! From getting pints to climbing trees, movie nights, and every form of mischief in between, you provided the emotional bedrock of my time in graduate school.

An equally sincere thank you goes to Will, Kim, Hope, Steve, Mimi, Peter, Morgan, Dominic, and the entire Bolte family. You've all provided me with amazing friendship, daring adventures, and a much-needed break from the "PhD grind." My time with all of you has always left me feeling refreshed, and I look forward to getting into even more antics post graduation.

Much appreciation goes to to Katia, Carlos, and Hamid for their repeated insight and advice over the last five years. Your patience and wisdom for my questions on systems, networks, theses, academia, industry, and careers has been greatly beneficial. I've been exceedingly grateful for your presence on the UCSC campus and hope to remain in touch.

A special thank you goes to JJ: you have been a phenomenal and amazing advisor, and consistently went above and beyond the call of duty in order to make sure I was progressing and succeeding in the path that I wanted for myself. I can't imagine doing grad school any differently, or getting my PhD advised by anyone else.

Finally, deepest love, gratitude, and appreciation goes to my family. You've been an unwavering rock of support throughout this entire foolhardy endeavor, and I love you all very much.

Chapter 1

Introduction

Internet applications today are primarily concerned with *content* and *services*. Applications can be loosely divided into producers and consumers, with producers (i.e. servers) making content and services available and consumers (i.e. clients) requesting said content and/or services from a producer's servers.

Despite the observation that content and services drive Internet traffic, neither are reflected in the identifiers, names, and routes used by the Internet architecture today. Instead, the core Internet protocols use IP addresses to indicate the network attachment point of a host, and port numbers to address a specific application process running inside a host. These two values, indicated as an {ip:port} tuple, are then used by applications to uniquely identify a corresponding application process to the operating system.

This discrepancy between an application or user's interest (i.e. content or services) and the addressing tuple used in the Internet (i.e. addresses and ports) means that a mapping from the interest to the tuple must be created before the application can send or receive data.

Figure 1.1 illustrates a well-known example of this process: a client requesting a webpage. Starting with a user-provided piece of named content, `http://www.example.com/thesis.pdf` in Step 1, the application first internally splits the URL into two components, the hostname and the path (Step 2). The application then provides this hostname to a Domain Name System (DNS) resolver (Step 3), which uses the DNS to map the hostname `www.example.com` to a set of IP addresses (Step

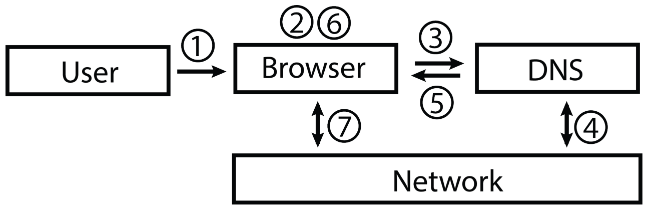


Figure 1.1: Traditional name-address resolution and binding

4) and returns this set to the application (Step 5). Next, following IANA standard convention[1], the application infers that HTTP traffic must be sent to port `tcp80` (Step 6). Finally, in Step 7, the application arbitrarily selects an IP address `addr1` from the returned set, opens a socket to `addr1:tcp80`, and sends a HTTP request to this socket.

Under this approach, the link-layer addresses and network routes between hosts are established and maintained in the system based on IP routing tables. However, the application is still individually responsible managing and mitigating several network-related concerns. These concerns include (1) parsing the hostname from the content name; (2) resolving the hostname to an IP address and binding it; (3) selecting the best IP address if more than one are returned; (4) discovering (or assuming) the correct port number and protocol; and (5) inferring that HTTP is the end-to-end content protocol used.

These existing algorithms for name resolution and address bindings have served us well since the inception of the traditional TCP/IP Internet [CK74]. However, as the Internet has become ubiquitous and wireless networks and devices have proliferated, new application requirements make the traditional approach to name resolution and name-address binding untenable. Specifically, supporting multi-homing and mobility of hosts and processes, seamlessly multiplexing among multiple network interfaces at each host, using diverse protocols in wireless networks, explicitly naming content or services, and evolving such identifying layers cannot be accomplished today under this model.

These problems and challenges are not new, rather, there exists a vast set of related work on adapting the Internet protocols and stack to better reflect the new

[1]The IANA port registry [ian] is simply assumed to be known *a priori* by the application.

realities of the Internet. This body of work ranges from abstract analysis on the nature of names, addresses and binding, to targeted works that focus on one specific network challenge (e.g. network address mobility or multihoming), to grand-vision “Future Internet” works that propose rearchitecting or redesigning the entire network stack and protocol suite. Unfortunately, these works come paired with a dramatically high adoption cost, generally requiring the complete redesign and replacement of Internet protocols, end-hosts, switches, and routers in the network.

To address all of these problems, the key contribution of this thesis is a new approach to the way names, addresses, and routes are bound between layers of the network stack. This approach enables applications to specify, and then bind, an identifier that semantically indicates *exactly* the name or resource requested by the application. The application then does not manage any other network-related concerns; these concerns include name resolution, address bindings, and routes. What is novel about our approach is that it specifically avoids introducing, requiring, or preventing the use of any new identifiers, services, or protocols in the Internet stack. This achieves support for several new network features (such as network mobility and content-centric binding) today while maintaining backwards compatibility, supporting future extensibility, and avoiding the roadblocks commonly associated with Future Internet architectures.

This thesis proceeds as follows. Chapter 2 summarizes and categorizes a vast set of related work in many different fields. Chapter 3 introduces the abstract concept of our approach, which we call *hidden identifiers*, explains the importance of hidden identifiers, and makes the argument for hidden identifiers in the network stack. Chapter 4 provides the design, implementation, and evaluation of HIDRA, the first network stack based entirely on hidden identifiers, and Chapter 5 introduces and evaluates DIME, a system that leverages the HIDRA architecture to support host mobility across IP address changes.

Going further, Chapters 6 and 7 show how HIDRA can be leveraged to support the goals of information-centric networking on top of the current TCP/IP network stack. Chapter 6 presents iDNS, a location system that extends the DNS infrastructure and protocol to lookup the current address(es) of content objects. Chapter 7 focus on supporting the goals of an information-centric data-plane (i.e. ubiquitous, opportunistic caching) by providing a method for adapting HTTP to support transparent content caching with blind security. Finally, Section 8 concludes the thesis.

Chapter 2

Related Work

The architecture of this thesis spans and integrates many different fields of work - as such, there exists a vast body of related work that we analyze, evaluate, and compare against. We divide this related work into several sections, in order: classic works on Internet identifiers (e.g. names, addresses, and routes), works that propose future or alternate Internet architectures (excluding Information-Centric Networks), work on Internet mobility and multihoming, work on information-centric networking, and work on content delivery and caching in the Web today.

2.1 Names, Addresses, and Routes

Work on the binding of names, addresses, and routes to one another goes back several decades. Watson [Wat81] provides an excellent summary of early work on the subject, and Shoch [Sho78] provided a famous characterization of these concepts: "the *name* of a resource indicates what we seek, an *address* indicates where it is, and a *route* tells how to get there." This set of primitives was further discussed by Saltzer [Sal93], who pointed out that an address is really just *the name of a lower-layer entity*, and the *binding* process connects a name to a particular address. It is implied that a particular layer in the network maintains and manages its named bindings to the next layer down.

Crucially, early works on the characterization of bindings among names, addresses and routes do not advocate how they should be carried out! Later works (i.e. [CK74] and the current TCP/IP network stack) simply assume a model wherein the names used by lower layers, and the binding of names to addresses enacted within a layer, are *exposed.* By exposed, we mean that these identifiers and their bindings propagate up the stack, and are visible/modifiable by higher layers (i.e. the application layer) as well as intermediate network entities (i.e. network routers and middleboxes).

Notably absent from prior work on names, addresses, and routes are works that challenge the assumption of exposed bindings or propose alternate models. However, such prior work can be seen in many other fields of computer science. Prior to the advent of computer networks, the original UNIX proposal [RT78] introduced file descriptors as a novel solution to the challenge of file location. Prior to UNIX, applications had to specify the exact location of a file on the disk or drum; consequently, any change to the operating system, underlying storage, or file system broke every application. In turn, file descriptors provided a standardized interface and *opaque* layer of abstraction that allowed applications to persist across such modifications by centralizing file location management in the operating system itself.

2.2 Future Internet Architectures

Several proposals have been made on how to evolve the Internet to address its current limitations with naming and addressing. FII [ea11b, ea11a] and Plutarch [et.03] highlight the fact that new solutions cannot be deployed incrementally, and must be uniformly adopted simultaneously. To address this problem, these proposals advocate for an Internet framework that allows for heterogeneity between different network domains (referred to as “contexts” in Plutarch). Both Plutarch and FII advocate a new network API, and FII suggests some guidelines for its design, but neither proposal provides a model of what this API should be, its implementation, or a roadmap for migrating applications to use it. Ghodsi et. al. [ea11a] briefly propose that a future network API should be based on hostnames, as opposed to network addresses. However, they describe the network API as something that must be redone from the ground-up, without specifics.

Other proposals [ea02, ea04, ea08, For08, ea12, Han12] advocate the introduction of new layers of transparent identifiers into the stack as a way of eliminating some of the naming and addressing problems in the current Internet architecture. In [ea04], the authors propose that applications start with a service identifier (SID) provided by the end-user, resolve it to a set of endpoint identifiers (EID), and then choose one to bind to a socket. EIDs are used only by the transport layer, and are translated and bound to network addresses in order for routing and communication to occur. A similar proposal, Serval [ea12], identifies the same problems and proposes the introduction of a Service Access Layer (SAL) between the network and transport layers. The SAL redoes the socket API to bind directly to service identifiers (SID) instead of the traditional tuple based on an IP address and a port number.

Other architectures [IMA10, TP08, DMM08, TGDM11, TBD$^+$11] explore the concept of recursion between layers of the stack. This model views each layer as providing an abstract interprocess communication (IPC) service to the layer directly above it, and thus views the entire stack as a recursive series of services that perform both transport *and* routing tasks, as opposed to a model where the entire stack constitutes one distributed IPC service for applications.

Unfortunately, these future network architectures all come at a very large adoption price. These architectures require the redesign of network applications and operating systems, and generally also require the replacement of all intermediate hardware (routers and switches). While some proposals [For08, ea12, et.03] lay out "deployment roadmaps" or explain how they can be incrementally deployed, many such proposals are incomplete or frought with concerning or inaccurate assumptions about what is feasible. As the most concrete example of this, we point to the adoption rate of IPv6: in January 2016, IPv6 reached a 10% connection rate *twenty years* after its standardization [ipv]! This rate is untenably slow, especially when it is taken into account that these future architectures propose much more radical network changes than the IPv4-v6 transition.

2.3 Internet Mobility and Multihoming

The core problem created by openly exposing identifiers and bindings is that higher layers inevitably ascribe separate and additional meanings to lower layer names. This problem is most evident in the treatment of IP addresses: 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. This semantic difference is famously known as the *identifier-locator split.*

This overloading of semantic meanings poses a significant challenge for persisting communication sessions across host address changes. Specifically, this challenge centers around the question of what to do when a host changes network attachment points: should the host *identifier* binding at higher layers be broken, should the host *locator* binding at lower layers be broken, or should the binding between these two functions be split? And if so, how should this be accomplished? Moreover, these problems are compounded by the ossification of each protocol in the network stack and the fundamental observation that if a name or address is bound, it must not be subsequently changed - such a change undermines the very purpose of binding!

While there exists a vast body of work addressing host mobility in the Internet, including several surveys [Edd04, PSS04, BP96, LFH06, KSB15, SMMC04, PJ96, GVK14, IMA10], we found that all approaches can be categorized by how they handle the identifier-locator split in the *data plane.* Specifically, we group prior work into approaches that transmit host identifiers, approaches that transmit host locators, and approaches that transmit both identifiers *and* locators.

2.3.1 Host-Identifier Approaches

The first host-mobility proposals, including Mobile IP [NB09, ABH09, Per97, KIUE00], preserve the identifier bindings made by higher layers, and approach host mobility entirely within the network layer, typically by updating a node's location in intermediate routing tables or rendezvous points as it moves, and using this information to redirect or reroute datagrams.

The key benefit of this approach 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) while still supporting host mobility in the network. Unfortunately, restricting host mobility to the network-layer creates significant problems. First, reliance on intermediate routers necessarily incurs additional control signaling and routing-table growth: host addresses cannot be aggregated if they are preserved across moves. Second, these approaches often increase network bandwidth consumption and end-to-end latency via triangle routing in the data plane. Finally, all approaches to network-layer mobility support require an infeasible amount of coordination, standardization, and replacement of existing infrastructure because they alter network-layer protocols in both the control- *and* data-planes.

2.3.2 Host-Locator Approaches

In response to the above problems, later proposals [Edd04, FRH$^+$11, SSII02, BS97, BB95, FYT97, FZ08, SB00, MB98, KG08, SGLA15] concentrate mobility support *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) by way of additional signaling or protocol options exercised when a host experiences mobility.

Host-locator approaches have seen more success than earlier works, because they do not require changes to the routing infrastructure and hence can be incrementally deployed at end hosts. However, these approaches are almost all TCP-specific and require significant protocol modifications. Additionally, they often introduce incompatibilities with NAT boxes and raise significant questions of scalability, because the migration protocol is enacted for every active connection at the host. Finally, implementing these modifications as kernel-level code renders these approaches remarkably non-portable, yet they cannot be implemented in user space without raising alternate questions of long-term compatibility and deployment [For08].

2.3.3 Combined Approaches

The most recent approaches transmit both identifiers *and* locators in the data plane, with the intent that higher layers only use identifiers and the network layer only uses locators. LISP and ILNP [NB09, ABH09] achieve this by splitting the IPv6 identifier space in half, though this raises significant concerns about address space fragmentation, introduces questions about the appropriate size of each value (lest one set of values run out), and requires ubiquitous IPv6 adoption.

Other approaches [SCFJ03, WS99, SW00, DVC$^+$01, PPKC06, KP04, ZM02, OMTT99, ea08, ea02, LCP$^+$05, Rip01, Coh08, CS05] identify hosts at the application layer through the use of overlay networks or rendezvous servers, and rely on application-layer shims for transmitting the identifier over an IP packet addressed to the locator. However, the reliance on overlay architectures incurs significant overhead in both the control and data planes, and can increase end-to-end latency to unacceptable amounts. Additionally, such distributed protocols can exhibit significant control message churn when nodes enter and leave the network. Finally, the increased network communication and resource consumption often renders these solutions inappropriate for resource- or bandwidth-constrained environments when compared to other more integrated solutions.

In addition to application-layer overlays, similar overlay networks can also be deployed by virtualizing and encapsulating either Layer 3 or Layer 2 traffic within UDP or IP packets [HF10, Mah14, Sri11, XY13, GHJ$^+$09]. When compared to application-layer approaches, these solutions are more adaptable and deployable, since they transparently support all existing network applications (e.g. email clients, web browsers, etc.) without modification.

Unfortunately, these system-based approaches incur even more overhead than application-layer approaches in multiple ways: (1) the end host must essentially route each datagram through the network stack twice, (2) doubling the headers incurs data-plane overhead that can exceed 50 bytes per packet; and (3) control plane overhead is *at least* doubled by the need to run virtual discovery and routing protocols on top of physical ones. These overhead costs, combined with the reliance on a complex virtual routing stack and IPSec encryption, renders these solutions expensive for traditional hosts (e.g., laptops) and infeasible for resource-limited or constrained devices.

2.3.4 Independent Host Namespaces

Other works [SB00, FZ08, RPB$^+$12, JSBM02, ea11a, ea10] propose resolving the identifier-locator split by entirely separating the layers and introducing a new layer or identifier namespace to uniquely refer to hosts. These protocols observe that applications typically identify a host through DNS resolution, rather than its IP address, and argue for either a socket API or TCP implementation based on *hostnames*, as opposed to network addresses. In this style, [ea04, MB98, ea08, KG08] advocate a similar model based on cryptographic host identifiers in place of hostnames.

Unfortunately, these proposals suffer from the same drawbacks of Section 2.2: they break backwards compatibility with corresponding hosts, network applications, and (in some cases) intermediate network hardware. Additionally, they require agreement on, standardization, and widespread deployment of new identity protocols.

2.4 Information Centric Networking

The research field of Information-centric networking (ICN) is motivated by the observation that Internet communication has shifted primarily to content distribution, Internet content continues to grow exponentially, and a large portion of this content is user-generated, to be shared with other users. This shift has prompted a substantial research effort [KCC$^+$07, JST$^+$09, FNTP12, ADM$^+$08] that proposes a "clean slate" redesign of the Internet architecture, shifting the focus from addressing *hosts* to denoting *content* in the form of named-data objects (NDOs). ICN proposals can generally viewed as future Internet architectures, but represent such a significant shift in network design and architecture that we consider them separately.

The various ICN proposals generally share a common set of *benefits*, namely: persistent and unique naming of data, efficient content-distribution, secure content provenance and authentication, and better support for network mobility, disruption, and multihoming. By the same token, ICN proposals also share a number of *characteristics*, including: content-routing based on NDOs, divorcing content names from location, ubiquitous content-caching at intermediate routers, and nearest-replica-routing [FLT$^+$13, ADI$^+$12].

Although the shift from addressing hosts to addressing content can provide many performance benefits in the future, the need to replace today's TCP/IP stack and routing infrastructure constitutes a significant obstacle to the adoption and deployment of ICN, given the ubiquity of IP routers and switches. Moreover, many open questions remain regarding naming, caching, routing, security, discovery, and scalability that must be solved before any one proposal is mature enough to be implemented at Internet scale.

2.5 Web Content Delivery and Caching

Independently of the ICN clean-slate research thrust, significant changes and improvements have been made to Web technologies over the last decade in order to better support content distribution. These approaches, which include Content Delivery Networks (CDNs), caches, and proxies, were designed to alleviate some of the problems associated with content distribution and dissemination, yet they have evolved in a very ad-hoc, disjointed manner. Though these technologies are not entirely information-centric, they implicitly support location-independent naming in that they serve the same data object from several locations. In this vein, HTTP itself can be loosely thought of as information-centric in that URLs name a piece of content [PGS10].

A recent set of "secure content delegation" proposals [TEH16a, RL16, TEH16b] that enable publishers to host content at a (presumably untrusted) CDN by providing clients with keys needed to verify and/or decrypt the content over a separate (out-of-band) HTTPS session. However, these approaches incur significant overhead (both in bandwidth and latency) by relying on redirection to obtain the location of the resource at the CDN. More importantly, by relying on redirection and requiring

HTTPS, the approach only supports explicitly configured CDNs and proxies, not transparent or opportunistic caches.

Meanwhile, policy-based solutions [LMS+14, Peo12, MWNG13] propose leaving TLS unchanged and "splitting" a TLS session into two separate connections: one from the client to the middlebox, and another from the middlebox to the server. However, since these proposals require the caching entities to be trusted either by root CAs and/or browsers, they completely break end-to-end authentication, and have thus met widespread resistance by the Web community [expc, expb, expa].

Finally, other works [NSV+15, SLPR15] propose altering TLS itself to support specific middlebox operations on traffic flows. However, these works appear to have focused exclusively on adding support for *qualitative* middlebox features such as intrusion-detection or content-filtering, and do not support transparent content caching. This limitation stems from the fact that such works enable middleboxes to alter existing TLS sessions, whereas the goal of transparent caches is to prevent an end-to-end flow from ever being established in the first place.

Chapter 3

The Design of Hidden Identifiers

As we summarized in Section 2, a large body of work has been aimed at making naming and addressing more responsive to the new realities of the Internet. From an abstract perspective, these proposals center around and focus on two key problems: (1) the "early binding" that must be established between the name of a process and the address where it can be provided, and (2) the need for applications to monitor and manage this binding.

Remarkably, all prior approaches address these two problems by introducing additional layers of identifiers into the protocol stack. While these proposed solutions appear attractive and semantically clean, we argue that these are *not* the correct approach to solving the naming and addressing problems of the current Internet architecture. This is because adding identifying layers to the stack effectively requires the development, standardization, and deployment of new communication protocols. Consequently, such approaches introduce significant roadblocks to adoption and deployment, including (1) requiring middlebox support; (2) replacing network-layer protocols and/or hardware; (3) significantly altering the operating system network stack; and/or (4) requiring modifications to network applications and the socket API.

In this chapter, we provide a new take on this old problem by introducing the concept of *open* and *hidden* identifiers and explaining the important difference between them. We discuss the challenges and problems with open identifiers, provide the core argument for opacity between layers of the network stack, and show why hidden identifiers are needed in the network stack. Finally, we provide rules and guidelines that ensure proper hidden identifier behavior, and provide examples that show what hidden identifier bindings would look like.

3.1 Open and Hidden Identifiers

3.1.1 Open Identifiers

A striking similarity of both today's Internet architecture and all the proposals in Section 2 is that they all, without exception, assume that protocols must employ what we call **open identifiers**. Open identifiers are transparent values that encode meaning, are propagated over a network, and are used to name or address[1] a network entity. Formally, we describe open identifiers with the set of three characteristics below.

- Open identifiers are visible to other network stack layers as well as the end systems or intermediate systems that employ them.
- Open identifiers are unique and unambiguous within a scope.
- Open identifiers cannot change once bound.

Examples of open identifiers employed today include DNS hostnames, transport layer ports, and IP and MAC addresses. Examples of transparent identifiers proposed by future Internet architectures include endpoint, host, application, content, and service identifiers.

The implicit assumption of open identifiers by network protocol designers is so ubiquitous that it has been argued [ea04] that the only way to break the early binding between a name and an address is to introduce an additional layer of (open) identifiers between them. The key drawback of this approach is that it still locks applications, protocols, and the network API to whatever new open identifiers are employed! This is a big problem for Internet evolution: just as the designers of the

[1]In this context, we use "name" and "address" interchangeably, since as [Sal93] points out, an "address" is just the "name" of a lower-level entity.

original Internet architecture could not predict today's problems associated with early bindings of names to addresses, it is not possible to predict what problems may result from the use of such new identifiers that must be unambiguous on a network-wide basis. Furthermore, requiring applications to use new identifiers in the API forces developers to modify applications and protocols as the Internet evolves.

3.1.2 Hidden Identifiers

In contrast to open identifiers, the primary contribution of this thesis is the abstract concept of **hidden identifiers**. Hidden identifiers are opaque values that explicitly encode no meaning and are not propagated among end systems or intermediate systems (hosts or routers) operating in a network. Formally, we describe the characteristics of hidden identifiers below.

- Hidden identifiers are mapped to one or more hidden or open identifiers via a table maintained in the operating system.
- Hidden identifiers completely mask the true value(s) and format(s) of the mapped open identifier from other protocols.
- Hidden identifiers are used internally by a layer or application in place of an open identifier.

An example of a hidden identifier might be the value `0x1`, when paired with a corresponding binding in the system that maps the hidden identifier `0x1` to the IPv4 address `192.168.0.1`.

Figure 3.1 provides a simple comparison between open and hidden identifiers. Figure 3.1(a) illustrates the Internet protocol stack today, in which TCP must work directly with open identifiers (the values ip1 and ip2). In contrast, Figure 3.1(b) shows how TCP would operate using hidden identifiers (the values hidX and hidY) without any knowledge of the corresponding open identifiers.

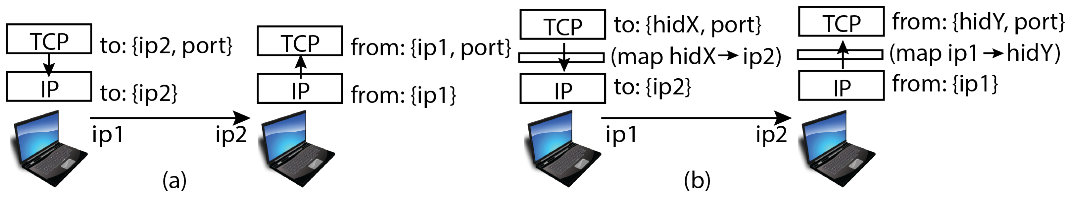


Figure 3.1: Open and hidden identifiers

3.2 The Arguments For Opacity

3.2.1 Identifiers, Locators, and Translation

[ea08, ea10, NB09, ea13] have each proposed a network stack architecture that uses a *host identifier*, such as a DNS hostname or HIP identity, in the socket API instead of an IP address. These works then adapt the socket API and/or transport layer such that both layers transparently bind the host identifier, and then the host identifier is translated to the network locator between the transport and network layers. Such architectures enable network-layer mobility, and have been well-received in the research community precisely because they leverage existing namespaces and infrastructure. Consequently, they do not require the development, agreement, standardization, and deployment of a separate identifier namespace.

However, this entire set of proposals suffers from two key drawbacks. First, the host identifier is still an *open* identifier, so the constraints of the chosen host identifier still apply. For example, DNS-based sockets cannot support environments where DNS is inappropriate, such as MANETs, and HIP-based sockets cannot support resource-constrained environments that do not support cryptography, such as RaspberryPis. Second, the architectural design of adding a new identity layer between the transport and network layer effectively breaks interoperability with middleboxes and other intermediate systems, and inhibits backwards compatibility and gradual adoption [For08]. Finally, since all of these works explicitly propose that the new namespace be deployed between the transport and network layers, they are implicitly bound to the TCP/IP stack. As a consequence of this, these proposals cannot support alternate network stacks that exist today, such as Bluetooth, Zigbee, or NFC, and are equally unable to support any future Internet stack proposals, including (1) service-centric proposals, (2) information-centric proposals, or (3) any other alternate network stack. Thus, we strongly argue *against* the injection of new naming layers into the existing data-plane, and argue that the proposals described here merely substitute one set of problems for another.

3.2.2 File Descriptors and Opacity

While hidden identifiers have not been used in any previous Internet architecture, they are not new to computing system design. Specifically, the design of hidden identifiers is very heavily influenced by the introduction of file descriptors in UNIX [RT78] as a way to provide a standard interface for applications that did not depend on either the physical location of the file or the underlying addressing scheme. Before the adoption of file descriptors, applications had to be written for specific hardware profiles, and this provided a significant roadblock to innovation, given that minor changes in the hardware broke all the applications. This problem is analogous to the state of network programming today, where changes in network addresses disrupt connectivity and changes in network protocols require applications to be rewritten.

By adopting file descriptors, applications remain ignorant of lower-level concerns, and this has enabled tremendous innovation in both filesystem and hardware design. Similarly, the use of hidden identifiers in the network stack provides an architectural solution to the majority of problems in today's networks by allowing different components of the stack to evolve and change independently of each other. In contrast, an API based on open identifiers is not nearly as modular: by design, an application using a open identifier must specify both the identifier and its format. This implicitly binds the application to whatever values were supplied, and ensures that the application must deal with any change in either value, such as switching addresses or protocols.

From the perspective of the application, hidden identifiers enable simple applications to take advantage of a wide range of network features. More importantly, applications written using hidden identifiers can automatically "opt-in" to new network features without being re-written. Application developers do not have to, and are specifically prohibited from, make any assumptions about network addresses, protocols, or stacks being used, or the features provided by the host stack.

3.2.3 Identifiers at Intermediate Systems

In network routing and forwarding, a resource or destination must be consistently denoted with the same identifier by all network forwarding devices (e.g., middleboxes, switches, and routers) in order for data packets to reach their intended destinations. Hence, from the perspective of the network, only open identifiers are useful.

The very nature of open identifiers (and, correspondingly, the very nature of forwarding and routing) requires that they be unique, unambiguous, and supported within the domain and scope in which they are to be used. For example, two hosts on the same network cannot share the private IP address `192.168.100.1` or multicast DNS name `name_1.local`.

Interestingly, open identifiers need not be global. Multiple types of open identifiers may be needed in the network, because globally unique identifiers may not make sense in certain networks (e.g., Plutarch [et.03]) and may be considered detrimental in others. For example, a network of things inside a house might prefer to only use local addresses for purposes of security, and an extremely resource-constrained sensor network may not be able to afford the overhead of a universal identifying protocol - even the IPv4 header today is considered overly bloated for sensors!

3.2.4 Identifiers at End Systems

Open identifiers have *also* been used to identify resources in end systems (hosts) in all Internet architectures, starting with the original proposal by Cerf and Khan [CK74]. At first glance, this appears to be a trivial choice, given that intermediate systems (routers, switches and middle boxes) require the use of open identifiers. However, this choice overlooks the fact that end systems manage resources individually, while intermediate systems only do so in coordination with other systems. More importantly, it ties host-centric protocols (i.e. the transport and application layers) to the specific protocols and formats of identifiers used in the network, which inhibits the deployment of any new networking approach based on new open identifiers.

To allow the applications and the Internet to evolve more freely, a host should be allowed to internally denote resources by means of hidden identifiers known only within the host, and translate the hidden identifiers it uses to open values before packets are sent out over the wire. Given that the actual values of hidden identifiers are meaningless until translated by the system, such identifiers can easily support multiple network stacks, protocols, and values simultaneously, as well as switching between them. Moreover, they can do so *without* requiring additional overhead, coordination, or a separate identity layer.

3.3 Hidden Identifier Acquisition and Semantic Binding

For hidden identifiers to be used in the network stack, they must first be created. Specifically, application processes must create a hidden identifier and *semantically bind* the hidden identifier to the exact network entity they desire. In this section we discuss the scope, concerns, and design of this semantic binding process.

3.3.1 Semantic Bindings at Applications

We started this thesis with the observation that network activity is driven by user applications, and that these applications typically desire services or content, not just to communicate with arbitrary {ip:port} tuples. It follows that a better network API should allow such applications to bind identifiers that represent exactly the resource desired by the application, yet not shoehorn applications into specific formats or require dramatic changes to the way such applications are written.

Any identifier that unambiguously encodes meaning is, by definition, an open identifier. It is, therefore, unavoidable that the identifier used by an application to identify its desired service or content must be an open identifier. However, the application need not, and we argue must not, bind any other open identifiers! It follows that in a hidden identifier architecture, the first step is to translate the *exact* open identifier requested by the application to a hidden identifier that the application can then use to send or receive messages.

```
struct sockaddr_hidden sa;
sa.hid = http_resolver("www.example.com/thesis");
sock = socket(AF_HIDDEN, SOCK_STREAM, 0);
connect(sock, sa, sizeof(sa));
/* regular socket communication here */
```

Figure 3.2: Hidden Identifier Application Pseudocode

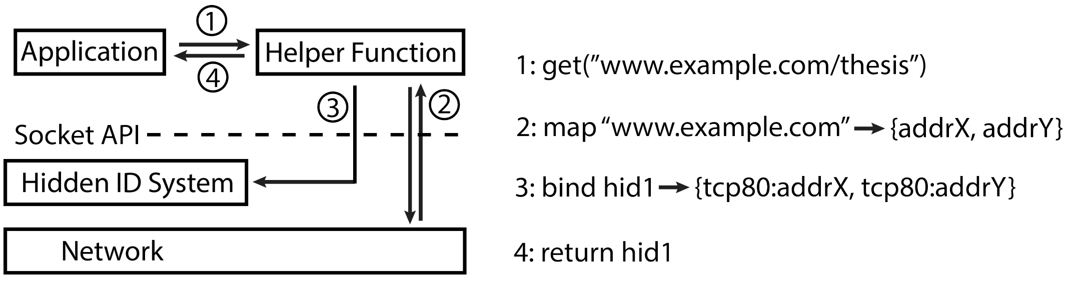

Application
Helper Function
①
④
③
②
Socket API
Hidden ID System
Network
1: get("www.example.com/thesis")
2: map "www.example.com" → {addrX, addrY}
3: bind hid1 → {tcp80:addrX, tcp80:addrY}
4: return hid1

Figure 3.3: Hidden Identifier Acquisition

To provide a tangible example of this binding process, we return to the use-case provided at the beginning of Chapter 1, that of a client loading a webpage. Semantically speaking, the client wishes to view the content object named `http://www.example.com/thesis.pdf`. Therefore, the client should start by semantically binding the open identifier `http://www.example.com/thesis.pdf` to a single hidden identifier that represents this object, and then open a socket directly to this hidden identifier. Figure 3.2 illustrates this process in pseudocode.

Figure 3.3 provides an architectural illustration of how such a hidden identifier can be created by a "helper" or library function. In this particular case, we already know that (1) the TCP/IP stack is the one to be used; (2) because HTTP is the named protocol, communication should be sent to port `tcp80`; and (3) the destination IP address can be *any* of the IP addresses returned by querying the DNS for `www.example.com`. It follows that all of these inferences and operations can be centralized in the helper function, and explicitly kept out of the application logic.

3.3.2 Peripheral Resolution Functions

A vital component of a hidden identifier architecture is that the initial mapping and acquisition of hidden identifiers take place through a peripheral resolution function and *not* the socket API itself. This split enables support for a much more diverse set of resolution protocols by specifically avoiding the standardization process inherent in an API.

In addition to host, service, and content identifiers, future Internet proposals have proposed attribute-based querying (e.g. `type=printer,loc=lab5`), implied scoping (the Bluetooth device currently paired with my computer and labeled as "My Phone"), or cryptographic identifiers (such as the Host Identity Tags employed by HIP [ea08]). While it would be exceedingly difficult, if not impossible, to design a single, unifying socket function that could support such a diverse set of input parameters, each individual protocol can easily be implemented as its own specific resolution function with its own parameters. Such functions can be written in userspace, can coexist with each other, and simply need to bind and returns a hidden identifier.

Building on this argument, even more approaches to hidden identifier acquisition could emerge, such as obtaining a hidden identifier from another application, via an IPC process or other specifically tailored helper function. These methods could allow for intricate relationships between applications to emerge, and could also provide additional security measures, such as providing an application a hidden identifier from a "black box" function without allowing the application to know what values the identifier multiplexes to.

The decision of generating hidden identifiers through peripheral resolution functions also draws inspiration from existing resolution functions: DNS resolution through the `getaddrinfo` function is implemented as a user-space library function, as opposed to a kernel-space syscall. This design keeps the complexity of DNS resolution out of the kernel, which contributes to system speed and stability by ensuring that the "minimal code path" remains lightweight.

By enforcing these restrictions, we achieve the significant benefit of lowering the bar needed to deploy a new resolution protocol. Creating or updating a function implemented in a user-space library is far less challenging or risky than a kernel-level change, and thus allows a vast set of different resolution protocols to be developed and distributed without compromising or affecting the operating system itself, or even interacting or depending on one another.

This design decision effectively decouples innovation in both resolution protocols and the network stack itself from network applications. Resolution and discovery functions can be introduced, modified, and updated without changing any other part of the socket API, and this minimizes the disruption to both applications and the system; such a degree of flexibility and extensibility is crucial when considered in the context of future Internet architectures and evolution. Continuing the example of loading a webpage, both the resolution function and the underlying network stack could undergo massive architectural changes, ranging from replacing the DNS resolution protocol with a DHT-based alternative to migrating the network protocol from IPv4 to IPv6, without requiring *any* modifications to the network application pseudocode in Figure 3.2!

3.4 Servers and Listening Applications

For an Internet application to *receive* messages in the traditional Internet architecture, it must accomplish two tasks: First, it must bind a socket to a particular network protocol and protocol-specific identifier, such as an IP-port tuple. Second, it must announce its presence by registering the identifier with a discovery or resolution service. Despite its vital importance, this second step is typically overlooked or executed in an ad-hoc manner, such as configuring a DNS server, manually distributing information out-of-band, or relying on a priori knowledge that certain ports correspond to certain services. The one exception to this claim is the mDNS API, which formally requires applications to programmatically label their services with user-friendly names before they can bind a listening socket to the service.

Because a hidden identifier architecture specifically masks the open network identifiers from applications, it must *mandate* the use of registration functions that complement the resolution functions described above. Following the mDNS model, applications wishing to receive datagrams must first use a peripheral function to register the resource or service they wish to provide. The application provides the peripheral function with an open identifier that exactly describes this resource or service, and receives a hidden identifier in return.

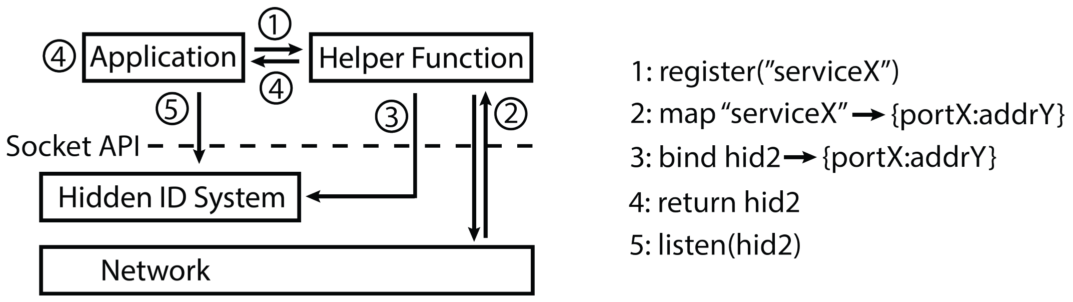


Figure 3.4: Service Registration and Binding

Just as in Section 3.3.2, the peripheral function is responsible for executing whatever registration or resolution operations are necessary, binding the appropriate open identifiers to a hidden identifier, and returning that hidden identifier to the application. This process is illustrated in Steps 1-4 of Figure 3.4. Once a service is registered and mapped to a hidden identifier, the application may then use it to bind a socket through the socket API the same way it binds a socket today, as shown in Step 5.

3.5 Hidden Identifiers in the Network Stack

If applications and protocols at an end host bind hidden identifiers instead of open ones, then the hidden identifiers must be carefully bound to ensure that they do not interfere with or disrupt connectivity.

3.5.1 Hidden Identifier Multiplexing

Once an application has acquired a hidden identifier through the acquisition process in Section 3.3, it uses this identifier to communicate with the standard socket API. When an application sends messages to a socket by calling `sendmsg`, instead of passing an open identifier (i.e. an IP address and port), the application passes the hidden identifier instead.

In turn, the system must translate the hidden identifier to a *complete set* of open identifiers. This is because, as we have discussed in Section 3.2, only open identifiers are useful for intermediate systems and protocols on the wire. We use the term complete set to mean a set of one or more open identifiers that is sufficient to successfully deliver communication in the data-plane. The number of open identifiers in a complete set, and the values of these identifiers, can and will change dramatically depending on the specific network context and protocol stack employed. An example of a complete set of open identifiers in the standard TCP/IP network stack is the {port, ip} tuple. In the case where a hidden identifier is bound to multiple open identifiers, or multiple complete sets, only a *single* set is to be chosen for transmission.

This translation occurs at a table maintained by the operating system; we call such a table the *Resource Descriptor Table* (RDT). This table must be centralized in the operating system itself, so that all issues pertaining to hidden identifier binding and updating can be managed at one specific location. This enables the operating system, which arguably has the best and most complete view of the state of the network, to make singular decisions as to what open identifier(s) should be chosen, and also ensures that application processes do not and cannot make such decisions on their own.

This design keeps applications simple, yet still enables them to take advantage of integrated support for features such as mobility and multihoming. It also enables policies to easily be implemented system-wide, such as preferring one protocol to another, load-balancing, or interface multiplexing. Moving this multiplexing to the system enables greater optimization and decision-making with a more complete perspective of the state of the network, and also enables the system to fully mask recoverable errors (e.g., routes changing, or one network address going down) from the application.

3.5.2 Demultiplexing and Scoping

For applications and protocols to receive and process messages properly and consistently, the inverse operation must be executed. When an end-host receives a datagram addressed to and from a complete set of open identifiers, it uses the RDT to demultiplex the open identifier(s) to the correct hidden identifier before delivering it to the appropriate application or protocol for processing.

This demultiplexing process must occur successfully, correctly, and unambiguously for each received datagram, otherwise datagrams will be incorrectly delivered to the wrong application process. For this demultiplexing process to work correctly, we note that the state of the higher layer must be maintained across any lower-layer changes. Therefore, hidden identifiers may only multiplex lower-layer open identifiers that are equivalent from the perspective of the higher layer. This effectively means that the state of the higher layer *includes* any identifiers, open or hidden, that it binds. As a result, enforcing this simple requirement ensures that the identifiers used by the higher layer are scoped correctly, and can therefore be resolved unambiguously by the higher layer.

To illustrate how such scoping influences demultiplexing, consider the case of transport-layer port numbers. The scope of a port number is the host in which the process resides; this scoping limitation ensures that the same port may be reused at every end host. However, this same scoping limitation *also* requires that a hidden identifier representing a port number must also be scoped by a host, since a port number out of such context is clearly insufficient to identify a process.

3.5.3 Connection-Oriented Protocols

Connection-oriented protocols typically provide abstract guarantees, such as reliable in-order delivery, to higher layers. Dynamically changing the addresses or protocols used can potentially violate these guarantees unless such a handoff is coordinated by the protocol itself. Thus, when a connection-oriented socket (indicated by the `SOCK_STREAM` argument) is bound to a hidden identifier, the system must be very careful as to if, when, and how it decides to multiplex the hidden identifier to multiple open identifiers.

Despite this constraint, connection-oriented protocols still benefit from the use of hidden identifiers, since they ensure that changes to a transport protocol do not propagate up or down the stack. For example, there exist several different proposals for TCP multihoming and mobility, ranging from opening multiple simultaneous TCP sessions [FRH+11] to implementing one of many solutions [BS97, FYT97, BB95] designed for in-flight handovers. These solutions are different architecturally, each has different advantages and disadvantages, and arguably more work will be forthcoming on transport-layer approaches aimed at handling mobility. However, from the perspective of the application and the rest of the network stack, all of these approaches are identical: the hidden identifier remains unchanged, and connectivity is preserved without application modifications.

3.6 Conclusion

In this chapter, we introduced the concept of *hidden identifiers*. We explained how they differ from *open identifiers*, and why this difference is important. We provided arguments for the use of hidden identifiers in the network stack, and showed that such a stack is feasible. Finally, we provided detailed rules that govern how hidden identifiers should be created and bound, and provided examples of how hidden identifiers can be used to replace some of the classic open identifier bindings in the stack.

Chapter 4

HIDRA: A Network Architecture Based on Hidden Identifiers

Based on the arguments and analysis in Chapter 3, we designed, implemented, and evaluated the first concrete network stack based on hidden identifiers; we call this network stack **HIDRA** (Hidden Identifiers for Demultiplexing and Resolution Architecture). In this chapter we describe the exact details and implementation of HIDRA, provide a comprehensive evaluation of its functionality as a network stack, and discuss several important use cases.

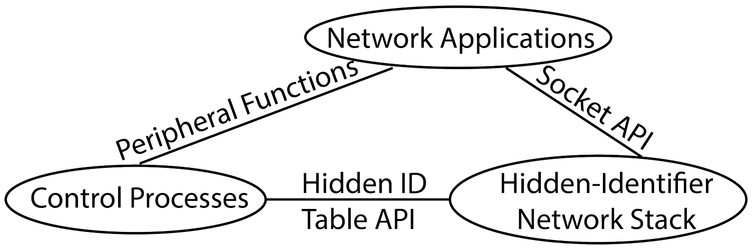
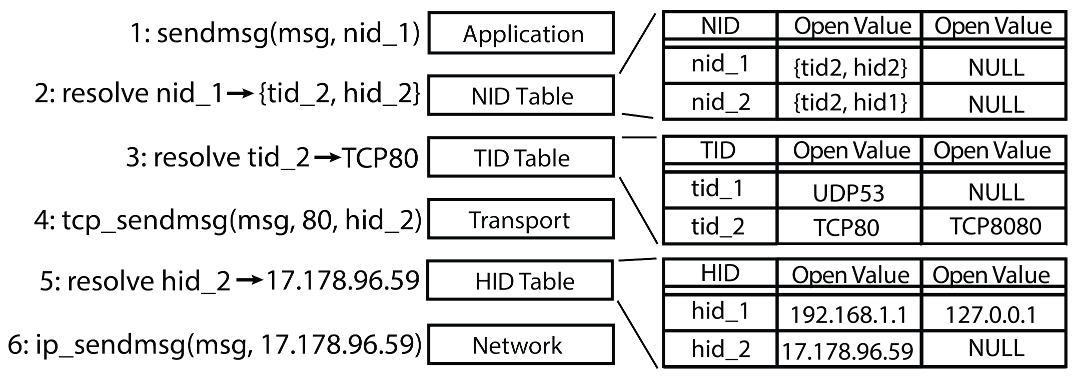


Figure 4.1: HIDRA overview

Figure 4.1 illustrates how HIDRA can be organized into three closely-related and interworking components, detailed in the following three sections. Section 4.1 describes the core design of how the protocol stack uses hidden identifiers between protocol layers, and how this process works when sending or receiving datagrams. Section 4.2 describes how network applications use peripheral functions to create an identifier that exactly represents the network resource they desire, and use this identifier to communicate through a protocol-agnostic socket API. Finally, Section 4.3


1: sendmsg(msg, nid_1)
2: resolve nid_1→{tid_2, hid_2}
3: resolve tid_2→TCP80
4: tcp_sendmsg(msg, 80, hid_2)
5: resolve hid_2→17.178.96.59
6: ip_sendmsg(msg, 17.178.96.59)
Application
NID Table
TID Table
Transport
HID Table
Network

NID	Open Value	Open Value
nid_1	{tid2, hid2}	NULL
nid_2	{tid2, hid1}	NULL

TID	Open Value	Open Value
tid_1	UDP53	NULL
tid_2	TCP80	TCP8080

HID	Open Value	Open Value
hid_1	192.168.1.1	127.0.0.1
hid_2	17.178.96.59	NULL

Figure 4.2: HIDRA protocol stack

explains how the mapping of hidden to open identifiers is created, maintained, and updated through control processes to reflect the original semantic binding requested by the application.

4.1 HIDRA Network Protocol Stack

4.1.1 HID, TID, and NID Semantics

HIDRA employs three sets of hidden identifiers: *Host Identifiers* (HIDs), *Transport Identifiers* (TIDs), and *Network Identifiers* (NIDs). Figure 4.2 provides examples and illustrates the position of these three identifiers in the protocol stack, and shows how all three hidden identifiers are maintained and multiplexed through tables.

A HID is a hidden identifier that sits between the transport and network layers, and maps to one or more network-layer open identifiers (i.e., IP addresses). Since the HID must preserve the state and scope of the transport layer, a single HID may multiplex across different network identifiers owned by the same host, but may *not* multiplex across different hosts. In this context, a "host" can refer to a physical computer, a virtual machine, or any such entity that maintains a transport-layer state. Additionally, HID multiplexing *across* hosts can still work if the transport-layer state is correspondingly migrated from one host to another. However, we leave such further discussions to future work.

A TID is a hidden identifier that sits above the transport layer and maps to one or more transport layer open identifiers (i.e., ports). Given that transport-layer identifiers are scoped to a particular host, TIDs are scoped to a particular HID for table storage and multiplexing. As it is the case with an HID, a single TID may multiplex across open identifiers just as long as the corresponding application-layer state is preserved. This enables application-layer services (i.e., a HTTP server) to dynamically bind and migrate ports.

From the perspective of the socket API, replacing a network address with a HID and a port with a TID masks the open values of these identifiers from the application using them. However, simply allowing applications to bind a {TID, HID} tuple is still problematic. This is because such a tuple still implies and requires certain restrictions of the underlying networking stack implementation. These restrictions are: (a) the existence of a transport and a network layer that use open identifiers; (b) the *lack* of any other such identifying layers (e.g., layers that identify services, hosts, or content); and (c) the need by the underlying network stack to use exactly two hidden identifiers. Furthermore, binding a socket to a {TID, HID} tuple ensures that the application is bound to exactly one TID and HID.

We address the above restrictions with the use of NIDs. A NID is a hidden identifier used by applications with the socket API. The NID is agnostic to any specific protocol stack or protocol, and is designed to mask *all* network stack logistics from the application. Thus, the NID can be multiplexed to one or more {TID, HID} tuples, a traditional {IP, port} tuple, a Bluetooth identifier, another NID, or any other such value, including but not limited to a set of one or more identifiers used by a future network architecture. How applications acquire and interact with NIDs is the subject of Section 4.2.

For organizational simplicity, in the remainder of this section we explicitly assume that the application has already obtained a NID that multiplexes to a valid TID and HID, and that the TID and HID correspond to valid open identifiers. The following two sections elaborate on how both of these points are achieved and maintained.

4.1.2 Connecting, Sending, and Receiving Messages

Steps 1-6 of Figure 4.2 illustrate how an application sends a message or connects to a NID.

First, the application passes a NID to the socket API instead of the traditional {IP, port} tuple (Step 1). The system multiplexes the NID to a {TID, HID} tuple (Step 2), translates the TID an open identifier (Step 3), then passes the message to the appropriate transport protocol. The transport protocol processes the message and creates a datagram addressed to the HID (Step 4). When the transport protocol is finished, the HID is translated to a open network address (Step 5), and the network layer processes the packet normally (Step 6). These same steps are taken whenever data are sent to the socket, regardless of whether the application calls `sendmsg()` to send a datagram to a NID, `connect()` to open a stream, or `send()` to send data to an established stream.

To receive messages, the inverse process occurs. After the network layer is done processing a packet destined for the host, the source network address is multiplexed to a HID. If no entry exists in the HID table, as can be the case for an incoming connection to a server, a new HID is generated. The transport layer then processes the packet and multiplexes the {port, hid} tuple to a TID. Note that as we discussed in Section 3.5.2, port numbers must be scoped by HIDs because ports are semantically scoped to a host and reused across machines. Finally, the resulting {TID, HID} tuple is mapped to a NID, and then the message is queued for delivery to the appropriate socket.

4.1.3 Transport-Layer Changes

As illustrated in Figure 4.2, the transport layer still uses its own open identifier (i.e. a port) but replaces the open network identifier with a HID. Thus, transport-layer protocols must be modified to index connections using HIDs instead of open identifiers. This modification takes place in two different ways. First, the foreign network address is replaced by a HID when storing or looking up connections. Second, the local network address is entirely *removed* from the lookup tuple. This is needed because, by definition, all packets received by the transport layer are destined to the local host, and a HID referring to the local host would by definition be the same across local network addresses. When exposed to different layers or bound to NIDs, the local host is denoted as HID 0.

These changes are all that is necessary to ensure successful protocol operation and datagram delivery. However, if the HID is multiplexed across multiple network addresses and routes, datagrams from the same HID may arrive out of order; this is known to negatively affect the performance of certain transport protocols, such as TCP. We address this problem through the inclusion of a small buffer that sits between the HID table and TCP to re-order packets when necessary. We do acknowledge that this represents a significant and fundamental problem, and further discuss the architectural challenge of out-of-order transport layer delivery in both Chapters 5 and 8.

4.2 Application-Layer Interface

One of the goals of hidden identifiers is to make network applications as simple as possible. Chapter 3 describes argues for applications to use a cleanly defined two-step process, as opposed to managing several implicit and explicit identifier bindings as they do today. HIDRA uses the peripheral function architecture described in Section 3.3 to map the explicitly named open identifiers provided by an application to a corresponding NID.

4.2.1 Existing Semantic Bindings

Semantically, the simplest way to assign meaning to a hidden identifier is to create a one-to-one mapping with an open identifier. Table 4.1 outlines a set of peripheral functions that provide this basic service, which enables applications to semantically bind raw IP addresses and ports, just as in the current TCP/IP stack.

These helper functions highlight the important semantic difference between an application binding an open identifier because it is exactly what the application desires semantically (i.e., a network utility that explicitly wishes to test the reachability

Function	Comments
generate_tid_tcp(portno)	Creates a TCP TID
generate_tid_udp(portno)	Creates a UDP TID
generate_hid_ipv4(ip_addr)	Creates an IPv4 HID
generate_hid_ipv6(ip6_addr)	Creates an IPv6 HID

Table 4.1: Peripheral functions

of a particular IPv4 address) or binding an open identifier because the architecture provides no other way for the application to express what is *actually* desired.

4.2.2 Future Semantic Mappings

In addition to the semantic bindings that exist today, hidden identifiers can also be semantically bound to a wide range of identifiers proposed by researchers. In this manner, HIDRA provides baked in API support, and a clean deployment roadmap, for any of the future Internet architectures enumerated in Section 2.2.

Endpoint-centric architectures that support host mobility across network addresses map very well to the HID table. HIDRA can support such an architecture by implementing a directory service or discovery protocol that maps the identifier to a set of network addresses and binds them to an HID.

Alternately, service-centric architectures focus on *application* mobility and replication across multiple hosts. While these architectures generally call for the introduction of one or more new naming layers to uniquely identify these services as they move, we note that the primary function of these layers is not to add end-to-end or intermediate functionality, but rather to mask mobility through the use of an unchanging identifier. This distinction is crucial, because hidden identifiers achieve the same goal by using a peripheral function to locate the service initially, and then sending control messages as the service migrates.

Because multiple peripheral functions may coexist with each other, HIDRA can support several diverse approaches to endpoint- and service-centricity simultaneously! This enables different approaches to evolve over time, without requiring significant modifications to applications or requiring agreement or consensus on a particular protocol or identifier format. Furthermore, it also enables endpoint-centric

ID Function	Comments
create_id(family, addr)	returns the hidden ID
delete_id(id)	delete a hidden ID
add_oid(id, open_id)	add open ID to a hidden ID's set
remove_oid(id, open_id)	remove open ID from a hidden ID's set
set_policy(id, policy)	set ID muxing policy

Table 4.2: Hidden-identifier table functions

applications to use an endpoint-centric architecture, and service-centric applications a service-centric architecture, in the same system!

4.3 HIDRA Control Processes

HIDRA intentionally and explicitly splits the multiplexing of hidden identifiers in the data path (described in Section 4.1) from the tasks of populating and maintaining these values in their respective tables. This architectural split enables two key benefits. First, diverse control processes that create and modify the bindings between hidden and open identifiers can coexist and even work together to aggregate many different forms of information. Second, these control processes can coordinate with the peripheral functions mentioned in Section 4.2 to support and maintain the semantic bindings requested by the application.

4.3.1 Basic Table Interface

Control processes interact with the NID, TID, and HID tables through the simple table-management interface illustrated in Table 4.2. While these functions and their implementation are largely self-evident, the purpose of the bottommost function, `set_policy`, is more abstract. In those cases in which a hidden identifier maps to more than one open identifier, control processes use `set_policy` to specify how the system should select an open identifier when sending data. Such policies include round-robin, always choosing a particular address or subnet when available, or weighting certain addresses more than others.

4.3.2 Mechanism and Policy

The table management interface is intentionally kept as simple as possible; this choice stems from the system design principle of separating *mechanism* from *policy.* In addition to being good engineering practice, this split keeps the table-interface operations lightweight and fast, and enables control policies to swiftly be designed, deployed, and automatically integrated into the existing data path.

This roadmap for deployment provides an attractive "third way" when contrasted with the two standard approaches of (a) breaking compatibility by injecting a new layer into the network stack, or (b) injecting additional complexity within a layer by overloading open identifiers or encoding a mapping between them. Rather, with HIDRA, complex and diverse semantic bindings and policies can be simply represented using the functions in Table 4.2. For example, current proposals for identifier mobility or multiplexing generally employ either an end-to-end or a publish-subscribe architecture, yet either architecture can be adapted to HIDRA by modifying them to exist as separate application processes that create and receive control messages, and then express the meaning of these messages through the functions in Table 4.2.

Implementing the control signaling this way enables different approaches to coexist and even integrate with each other! For example, publish-subscribe architectures [ea10, ea08, P. 02, SMGLA13] must generally provide some mechanism to ensure that already-established connections are updated as identifiers move. However, given that hidden-identifier tables provide a unifying point for different control processes, such a goal could be accomplished through an entirely separate end-to-end signaling protocol.

4.4 Evaluation And Case Studies

We implemented HIDRA as a Loadable Kernel Module (LKM) for Linux 3.13.x. Linux 3.13.x was chosen because it is the base distribution for Ubuntu 14.04 LTS, Mint 17, and the current distribution of Raspbian. Our kernel module consists of a basic HIDRA socket API, NID, TID, and HID tables, as well as the table-management interface described in Section 4.3.

Building on this LKM, we also implemented several different peripheral functions to provide robust functionality for HIDRA applications. These functions include one that maps DNS host names to HIDs, one that maps service-protocol names to TIDs, and one that maps a cryptographic "shared secret" known by an application to a particular TID and HID. Finally, we used these tools to run a series of "case studies" that underscore and evaluate the flexibility, performance, and modularity of the HIDRA protocol stack, in terms of (a) writing HIDRA applications and control processes, (b) porting existing network applications, and (c) supporting new networking paradigms.

The evaluations in this section focus primarily on the application interface and performance of the network stack itself in the data plane. The experiments in this section explicitly do *not* evaluate the overhead and performance of HIDRA control processes or peripheral functions - these are the subjects of Chapters 5 and 6, respectively.

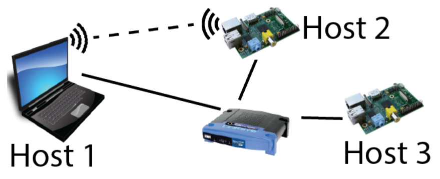


Figure 4.3: Testbed topology

4.4.1 Data-Plane Address and Host Multiplexing

For our first case study, we wrote a HIDRA `netcat` application, called `nc-hidra`, which supports both stream- and datagram-based communication. We deployed this application across one laptop and two Raspberry Pis as shown in Figure 4.3, and registered the set of network addresses of each computer at a local DNS server. We then configured the hidden-identifier tables at Host 1, such that an individual NID (used by nc-hidra) indexed two HIDs (referring to Hosts 2 and 3, respectively), and the HID referring to Host 2 indexed both of its network addresses. Finally, we connected a webcam to Host 1, and used nc-hidra to send datagrams from this

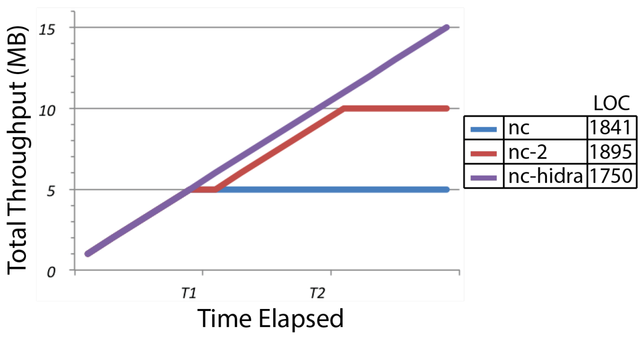


Figure 4.4: Netcat comparison

webcam to this NID. At time T1, we disconnected Host 2 from the 802.11 ad-hoc network, and at time T2 we disconnected Host 2 from the ethernet network.

With this configuration in place, we compared the performance of `nc-hidra` to unmodified `nc`, as well as `nc-2`, which we modified to be more resilient in the face of disruptions by storing all resolved network addresses for a host and reconnecting if possible. Figure 4.4 illustrates the performance of all three versions of `nc`, measured both in throughput received and total lines of application code.

At time T1, standard `nc` fails, but `nc-2` shows that extra application code can mitigate this failure with minimal disruption. However, because each host has a different local DNS entry, even `nc-2` is unable to multiplex across hosts and mitigate the complete disconnect seen at time T2. In contrast, `nc-hidra` uses HID multiplexing to mitigate the first disconnection without any loss in throughput, and NID multiplexing to mitigate the second disconnection just as easily.

4.4.2 Adapting Non-HIDRA Applications

In addition to being the only version of the application to persist across all forms of identifier changes, Figure 4.4 also reveals that `nc-hidra` requires the least lines of code! This is because all network-related handling is baked into the system itself, as opposed to the network application.

Exploring this point further, we adapted several traditional network applications to use HIDRA and measured the lines of code changed and the total number of lines of code. Our results are shown in Table 4.3, and show that adapting traditional applications to use HIDRA can be accomplished with minimal changes, which typi-

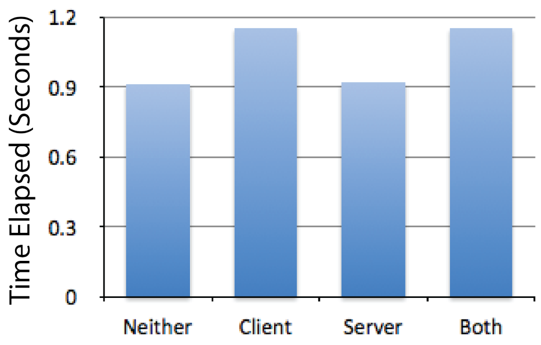


Figure 4.5: Hidratunnel overhead

cally required between 45 minutes and one hour. In addition, these results also show that in all cases, the HIDRA application is simpler and requires fewer lines of code overall.

Program	Lines Changed	Time Needed	Total Difference
nc	135	0:45	-91
iperf	333	1:15	-288
tftp	119	0:55	-73

Table 4.3: Lines of code

4.4.3 Legacy Application Support

Building on the above study, we also explored what is possible when the source code of an application cannot be made HIDRA-aware. This may be the case for many proprietary applications, especially those that are not frequently updated or those that have been completely abandoned.

To support these applications, we wrote a simple tunneling proxy application, which we call `hidratunnel`. `hidratunnel` supports datagram- and stream-based communication, both client- and server-mode, and works by tunneling a locally-bound INET socket to a foreign-bound HIDRA socket. Thus, by redirecting the unmodified traffic from the application through `hidratunnel`, the local connection is mapped to a HIDRA NID, and correspondingly receives all the benefits of the HIDRA protocol stack.

After developing `hidratunnel`, we deployed it with unmodified Firefox on Host 1, unmodified Apache on Host 2, and then timed a 1MB HTTP file transfer 4 separate

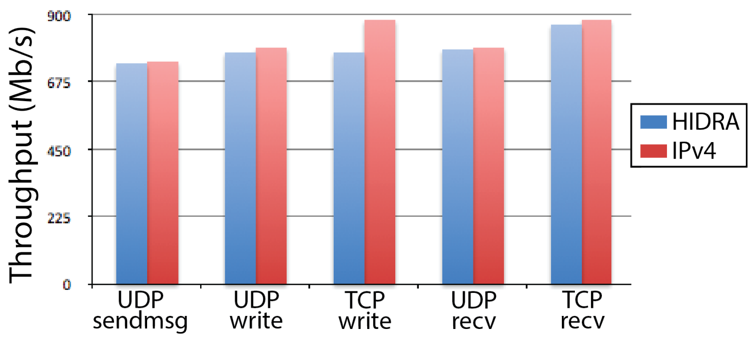


Figure 4.6: Multiplexing overhead

times: once over regular IP, once with `hidratunnel` at either side, and once with `hidratunnel` at both sides. Figure 4.5 provides these results, which show that `hidratunnel` does not incur significant overhead when compared to an un-tunneled connection.

4.4.4 Multiplexing Overhead

The per-datagram identifier multiplexing in HIDRA naturally incurs some performance overhead. To measure this overhead, we ran `hidra-iperf` over the loopback interface - this test effectively measures the performance and speed of the network stack itself. We tested three different socket API calls: write() requires the socket to have already been connected, sendmsg() requires an unconnected socket (therefore TCP does not support it), and recv() supports both states.

The results of our tests are summarized in Figure 4.6, and show that across all experiments the difference between HIDRA and IPv4 was consistently small, typically within 10 percent of the base stack. More importantly, the speed of the HIDRA protocol stack is still much higher than most network links, so it does not constitute a bottleneck when compared to other parts of the network.

4.5 Conclusion

In this chapter, we showed how the abstract design philosophy in Chapter 3 can be translated to an actual network stack implementation, which we call HIDRA. We introduced the three-part HIDRA architecture, and described how the combination of hidden identifiers, control processes, and peripheral functions works together to create a powerful and modular network stack architecture. Finally, evaluations of this network stack architecture show that (1) HIDRA-based applications are remarkably simpler than traditional network stack applications; (2) existing network applications can be easily adapted to HIDRA; (3) multiplexing hidden to open identifiers in the network stack does not incur a significant performance penalty; and (4) even a simple round-robin link multiplexing policy results in increased aggregate throughput at network links.

Chapter 5

DIME: Lightweight and Deployable Mobility at End Hosts

The HIDRA architecture described in Chapter 4 is a radically new approach to the way identifiers are bound, transmitted, and updated within the network stack. HIDRA provides a strong *foundation* for features such as network mobility and multihoming to be implemented as out-of-band control processes. However, HIDRA itself does not support these features, and such a mobility protocol must be carefully constructed in order to ensure proper operation in all network scenarios while avoiding edge-cases and deadlocks.

Building on our implemented HIDRA architecture, we constructed the first out-of-band mobility signaling protocol that leverages hidden identifiers in the network stack. We call this protocol **IHMP** (Internet Host Mobility Protocol), and call the combination of HIDRA and IHMP **DIME** (Deployable Internet Mobility and End-hosts). DIME is the first solution to Internet host mobility that can be seamlessly deployed at end hosts - that is to say, DIME consists entirely of userspace processes and does not depend on any further kernel-level or system-specific modifications beyond the HIDRA network stack. In this chapter, we explain the factors motivating and driving the design of DIME, identify key requirements that a mobility solution must exhibit in order to be deployable, explain the IHMP protocol design, and provide the results of extensive evaluations that compare DIME to all other implemented host mobility solutions.

5.1 Protocol Challenges and Requirements

Despite the relative maturity of many mobility proposals (more than 10 years old), no one proposal has seen much traction towards deployment. We analyzed the reasons why this prior work has failed and identified the following three key requirements that a mobility solution must exhibit in order to be feasible, implementable, and adoptable in the Internet today.

R1 - Unmodified Routing Infrastructure: Solutions that require modifications to intermediate network-layer hardware (i.e., switches, routers, or middleboxes) or protocols (e.g., IP, ARP, or BGP) will necessarily incur massive upgrade costs, and as such be met with widespread resistance by network operators. This creates an insurmountable hurdle, since any network-layer solution will also require adoption by a majority of operators before it could be reliably workable. It follows that a deployable host mobility architecture *must* run over existing routers and switches without requiring their modification or replacement.

R2 - Unmodified OS and Applications: The end-host mobility solutions in Section 2.3.2 all require substantial modifications to the system kernel. However, both proprietary and open-source system developers and maintainers consistently oppose such proposals [For08]. This opposition is grounded in the observation that these proposals dramatically increase code complexity, yet host mobility support is not a pressing or "tentpole" feature. Moreover, since such a feature will not be compatible with other systems that have not yet implemented it, the incentive to be the first to implement and support a mobility protocol is very low. In light of this opposition, the path to deployment for any mobility solution must be incrementally deployable, and specifically not depend upon buy-in from or modifications to applications, the host OS, or other system components.

R3 - No New Namespace: Many mobility solutions The majority of the proposals in Section 2.3.3 require the convergence on, and standardization of, a new host identifier namespace to be inserted into the network stack.[1] This represents a

[1]While many works propose leveraging the DNS as an existing namespace; [STU$^+$14] makes a strong argument for why the DNS is fundamentally unable to support host address mobility.

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 improbability of such an event is combined with our significant concerns regarding identifier lock-in and incremental evolvability. It follows that for a host mobility solution to evolve organically and incrementally, it must not depend on the development and standardization of a new identifier namespace.

Interestingly, the above three requirements loosely map to the architectural categorization provided in Section 2.3. Host-identifier approaches for host-mobility support typically satisfy requirements R2 and R3 at the expense of R1. Host-locator approaches satisfy R1 and R3 at the expense of R2, and combined approaches (Section 2.3.3) satisfy R1 and (sometimes) R2 at the expense of R3.

5.2 The Internet Host Mobility Protocol

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

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. We use UDP over TCP in order to stay lightweight; this also allows IHMP to interpret dropped messages as a sign that a path may not be sufficiently reliable. We use UDP instead of network-layer ICMP to enable NAT detection and traversal. When IHMP receives a message from a foreign host, it updates the HID Table to reflect the changes, at which point the HIDRA stack immediately incorporates them into the data path.

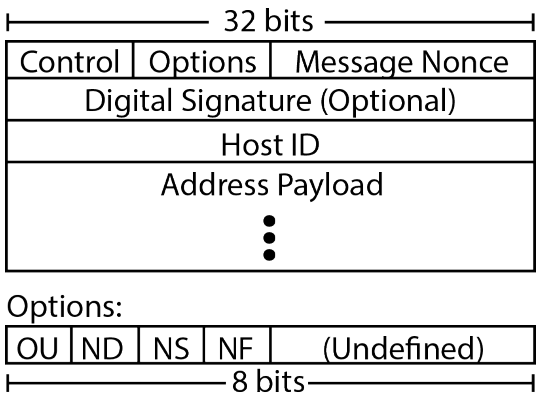


Figure 5.1: IHMP Message Format

Figure 5.1 illustrates the IHMP message format, and Table 5.1 lists the different IHMP message types. The control field corresponds to the message type in Figure 5.1, and the Message Nonce is a randomly-generated 16-bit value echoed by a responding `ACK`. A sending host may elect to append a digital signature to the message, but always identifies itself to the receiver via a local IP address stored in the Host ID field.

5.2.1 End-To-End Host Identification

The Host ID field in IHMP represents a major departure from all other proposals [NB09, ABH09, ea08, ea10, MB98] that use end-to-end updates. Whereas all prior proposals rely on a separate host identifier namespace (e.g., a DNS hostname or HIT) to consistently identify a host across network address changes, IHMP is specifically designed *not* to rely on or assume any such namespace.

This lack of namespace reliance or lock-in makes IHMP much more lightweight and deployable across a wider range of networks, however, it 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 receiver's HID; 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.

Crucially, this solution does *not* require globally unique IP addresses! Rather, it

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 5.1: IHMP Messages

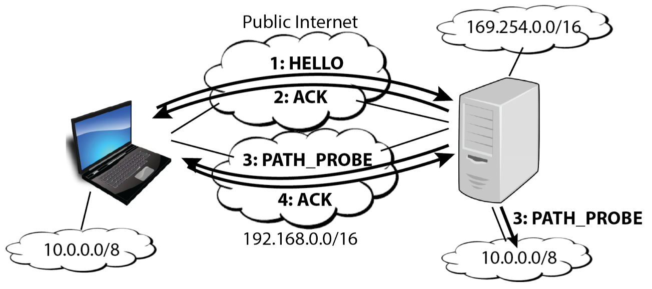


Figure 5.2: HELLO message exchange

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. This point is important, because it enables IHMP to support many challenging edge-cases where reachability is not flat or uniform, including communication within reusable (i.e. private) address spaces or communication through multiple NATs. This highlights the strength of IHMP over rendezvous or DHT architectures, as these designs implicitly require uniform, flat reachability between all nodes.

5.2.2 IHMP Hello Exchange

When an active host creates a new HID table entry, it must test the foreign host for (1) IHMP support and (2) the presence of other network addresses; it must also advertise its set of local addresses to the foreign host. The active host accomplishes this via a simple two-way `HELLO` message exchange, illustrated in Steps 1 and 2 of Figure 5.2. The first `HELLO` message contains all the network addresses the active

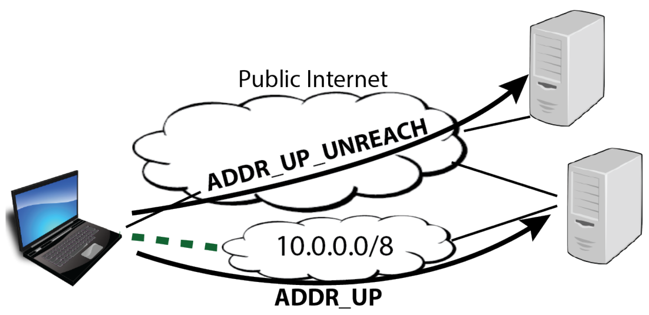


Figure 5.3: Address-up signaling

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. Next, it sends an ACK back to the active host at the same address used to send the HELLO; this 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.

5.2.3 Backpath Probing

From the perspective of the foreign host, the other addresses included in a HELLO message are either (1) guaranteed reachable (e.g., a publicly reachable IP address), (2) guaranteed unreachable (e.g., in a network that the host cannot reach), or (3) *potentially* reachable (e.g., a private network that the host also has an address in). The foreign host sorts the addresses in cases (1) and (2) into active and unreachable addresses, respectively, and manages the addresses in case (3) by sending a PATH_PROBE message as illustrated in Step 3 of Figure 5.2.

In our example, the active host initially reaches the foreign host over the public Internet, and indicates (in the HELLO message) that it also has addresses in the 192.168.0.0/16 and 10.0.0.0/8 networks. Subsequently, the foreign 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. When the active host receives a PATH_PROBE message, it updates the HID's set of active and local addresses to reflect reachability, and replies along the same path with an ACK (Step 4 of Figure 5.2); upon receipt of the ACK the foreign host does the same.

5.2.4 Address Up and 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. This explicit address encoding allows us to detect and mitigate NATs; we discuss this process further in Section 5.3.4. 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 address; the intent of this address is to ensure that the foreign host's HID table is accurately updated. As above, 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. In the case where 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.

5.2.5 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

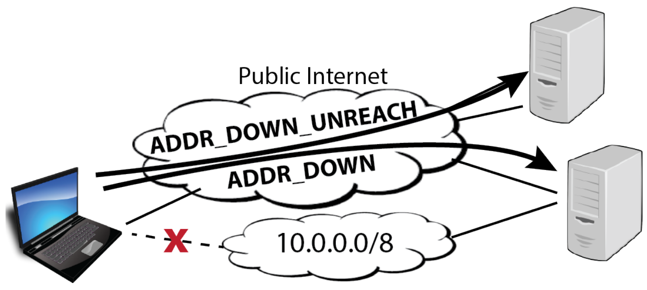


Figure 5.4: Address-down signaling

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.

5.2.6 Control-Plane Security

Since spoofed IHMP messages have the potential to disrupt and redirect communication, IHMP uses optional digital signatures to ensure message *integrity* without *confidentiality*. This decision reduces computational overhead and allows intermediate nodes (i.e., middleboxes) to (1) read the content of IHMP messages and (2) generate control messages signed with their own key without violating the security model; we discuss these optimizations further in Section 5.3.3.

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 discussions are well outside the scope of this work, except to mention that (1) IHMP is compatible with all public-key solutions and (2) IHMP explicitly does not *depend* on such a solution and is deployable even in network scenarios that do not require or support security protocols.

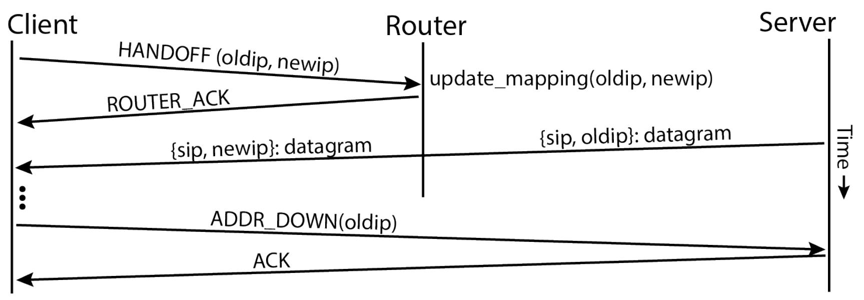


Figure 5.5: Micro-Mobility Signaling

5.2.7 Out of Band Signaling

The key differentiator of IHMP, when compared to other mobility signaling protocols, is that IHMP is the first and only protocol that exists completely independently of both the routing infrastructure as well as the communication data-plane. We have argued that the first point is vital for incremental deployment at end hosts, yet the second point is equally important. By removing mobility signaling from data-plane headers and protocols, we avoid the implementation problems of [For08] and enable seamless backwards compatibility with legacy hosts. Additionally, we achieve an architecturally pure identifier/locator split without requiring an extra layer of identifier headers in the stack or scaling with each active connection in the system. Finally, such a separation is the *only* way to consistently identify hosts across IP address changes without requiring the standardization and deployment of a separate host identifier namespace.

5.3 Additional Considerations and Edge-Cases

5.3.1 Simultaneous Mobility

DIME's lack of rendezvous nodes or integration with a name-resolution service poses a problem for *simultaneous mobility* cases, wherein both nodes change network addresses before the end to end signaling 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 will re-send the same mobility message to the host's other addresses until it receives an ACK from the address that was maintained; the partially-mobile host will then transmit 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.

5.3.2 Mistaken Identities

Since 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 gains A's old address $a1$, and (4) B sends a message from $a1$ to host X.

DIME guards against this scenario by verifying that the sequence number is correct before processing the update, and 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 this 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; if the network environment is considered insecure or hostile, digital signatures should be used.

5.3.3 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 can 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

(1) enact micro-mobility by updating or installing a new routing-table entry, as in Figure 5.5, or (2) indicate that the update is to be rejected.

Designating a separate ROUTER_ACK message type acknowledges middleboxes and routers as first-class citizens in 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 foreign host's HID table still contains the old address. This is illustrated at the bottom of Figure 5.5, where the host loses newip but sends an ADDR_DOWN message containing oldip.

5.3.4 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 L4 port renumbering. Since NATs are remarkably present in the Internet today and expected to be "here to stay" for the foreseeable future [For07], DIME must provide a mechanism for NAT detection and traversal in all cases, even those where 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 5.6. 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 be-

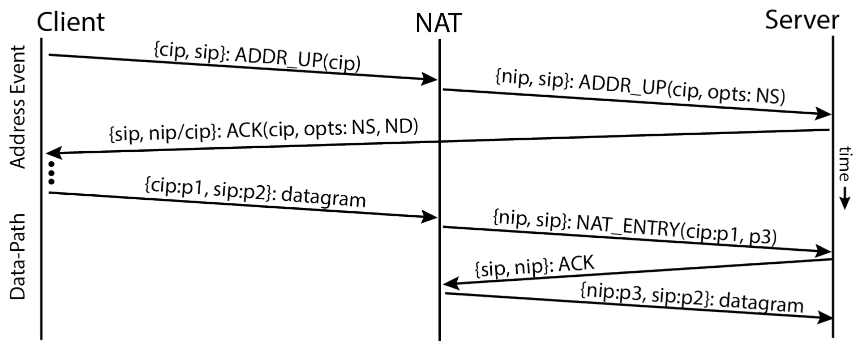

Client
NAT
Server
Address Event
Data-Path
time
{cip, sip}: ADDR_UP(cip)
{nip, sip}: ADDR_UP(cip, opts: NS)
{sip, nip/cip}: ACK(cip, opts: NS, ND)
{cip:p1, sip:p2}: datagram
{nip, sip}: NAT_ENTRY(cip:p1, p3)
{sip, nip}: ACK
{nip:p3, sip:p2}: datagram

Figure 5.6: Mobility signaling with NAT

hind 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).[2]

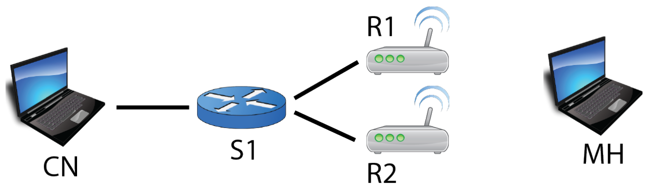


Figure 5.7: Testbed topology

5.4 Implementation and Evaluation

DIME exists as a combination of a userspace IHMP daemon and the HIDRA Loadable Kernel Module (LKM). We deployed this 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 5.7. 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 [PB15, ver, att, pin].

To evaluate and compare performance, we conducted a standard mobility experiment in which the Mobile Host hands off a TCP[3] connection from R1 to R2 while

[2]There potentially exist other techniques to mitigate the ID/Locator split across NATs. For brevity and focus, we omit further discussion in this paper and leave it as a promising topic for future work.

[3]While DIME supports UDP, we provide only TCP results (unless otherwise noted) in order to provide a fair comparison with MPTCP.

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

Table 5.2: Deployment Requirements

conducting a throughput test to the Corresponding Host. 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 compared DIME against three protocols, Mobile IP (MIPv6), Multipath TCP (MPTCP), and the Host Identity Protocol (HIP), which we chose as "flagship" examples to represent each of the three categories listed in Section 2.3.

5.4.1 Deployment and Configuration

We found that the different protocols varied wildly in terms of how much effort it took to configure even the basic testbed in Figure 5.7. Table 5.2 provides a rough summary of these protocols, and shows that 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 userspace daemon at all 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.

Installing and configuring MPTCP was much easier: while MPTCP requires a custom kernel at end hosts, it does *not* require any userspace daemons or configuration files, and is still actively maintained by a small developer community[4]. However, MPTCP's reliance on link selection by source address binding forces an unorthodox system configuration, wherein separate routing tables are maintained for each interface - while a minor point in our testbed, this raised the question of

[4]http://mptcp.org

MPTCP's ability to support more unorthodox or dynamic network configurations, such as those that rely on virtual network interfaces.

HIP deployment was even easier, in that the current HIP implementation is stable and deployable as a standalone userspace daemon that encapsulates packets via TUN/TAP. However, proper HIP configuration relies heavily on manually encoding static, preconfigured Host Identity Tag (HIT) bindings at both end hosts. Additionally, HIP's use of the 1.0.0.0/8 block for LSI bindings raised questions about HIP's support for other applications that depend on dynamically resolved or hard-coded IP addresses, as opposed to those that transparently take IP addresses as input.

Finally, we found that DIME "simply works." Similar to HIP, DIME exists as a standalone userspace daemon deployable on top of a stock kernel. DIME requires no network layer hardware or protocol modifications, and supports precompiled application binaries without modification. DIME's use of existing IP address bindings enables it to work without the need for specific configuration files, namespace bindings, or "pseudo" IP addresses, and DIME's use of translation instead of encapsulation makes it the only approach that can dynamically support preexisting connections.

5.4.2 Handoff Latency

After configuring the testbed, we evaluated the latency needed to complete a handoff, measured in two metrics: 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 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.* For brevity, and to enable a clear comparison with MPTCP, Figure 5.8 presents only TCP results collected over a hard handoff. However, we note that soft handoff results were effectively identical in all cases, and UDP results were comparable for every protocol that supported it.

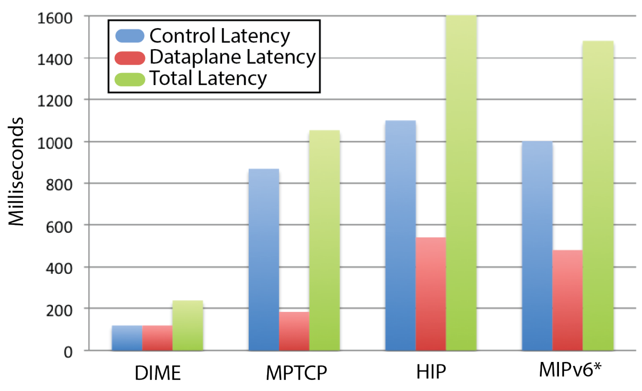


Figure 5.8: Handoff Signaling Latency

Control Latency

The results in Figure 5.8 show that in both types of latency, DIME dramatically outperforms all other approaches. With regards to control latency, despite significant examination, it was unclear exactly why both MPTCP and HIP wait so long (approximately one second) to start the mobility process, but it appears that the waiting period stems from a combination of code complexity as well as sending the first message too soon (i.e., before the interface is actually ready) followed by waiting before retransmission. DIME avoids such problems by existing in userspace - as such, it only receives address notifications once the local interface has been completely configured, and the datagram transmission itself buffers and triggers any necessary control messages, (e.g., ARP resolution). Additionally, by failing gracefully in the event of incorrect network configuration or incomplete signaling exchange, DIME can be more aggressive with its signaling.

The asterisk in Figure 5.8 indicates that the control latency of MIPv6 varied dramatically based on abstract timing as well as router configuration, since the mobility process is triggered *not* by when the host acquires a new network address, but by when the host receives a router advertisement (RA) message from the new router. In the interest of providing a competitive comparison we used the minimum RA interval of one second, and provided the best observed value instead of the average, but we stress that these conditions are (1) remarkably noisy and (2) very unlikely to be used in an actual network scenario.

Dataplane Latency

Since 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, it is not influenced by control latency. We found that our results for dataplane latency in Figure 5.8 are relatively self-explanatory, in that they very closely tracked the expected RTTs needed for the mobility signaling protocol to complete - this highlights the simpler handshake of DIME, as well as the relative lack of cryptographic operations needed to migrate the connection.

Connection Establishment Latency

We also examined the additional latency (if any) required by the protocol to *setup* a mobile connection versus a traditional one. This measurement, typically dominated by RTTs, has quickly become an important performance consideration as bandwidth increases and network traffic is increasingly driven by small communication sessions [RCC+11]. We found that in all cases except HIP, the connection establishment latency was minimal (<4ms), whereas HIP incurred dramatic connection establishment latency (400+ms for UDP and 800+ms for TCP). This comparison stems from the observation that all other proposals initialize mobility support either via out-of-band signaling or by piggybacking options in the TCP handshake, whereas HIP effectively requires a four-way-handshake to establish the HIP session before datagrams can be sent or received. Moreover, this session establishment process does not optimize with TCP and requires yet *another* three-way-handshake to be enacted for TCP sessions.

5.4.3 Data Plane Throughput

Since a mobility solution must not negatively impact the data-plane, Figure 5.9 provides the application-layer goodput seen over a fifteen second throughput test. We examine goodput, instead of link throughput, in order to provide a singly unifying metric that accurately reflects the performance seen by network applications and accounts for the many factors that affect different mobility solutions.

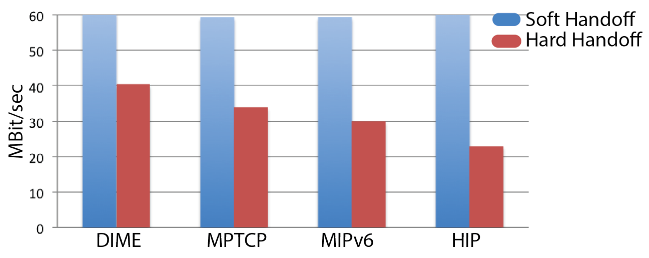


Figure 5.9: TCP Handoff Goodput

Figure 5.9 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 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 value for the hard handoff scenario will be approximately 2/3 of the soft handoff. Figure 5.9 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. In contrast, we found that MPTCP's performance suffered from the additional latency studied in Section 5.4.2, as well as the slow-start algorithm - while MIPv6 has a longer handoff procedure, it mitigates this with respect to goodput by migrating the existing TCP connection without triggering congestion-control. Finally, we found that HIP's remarkable decrease in goodput was 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.

5.4.4 Multipath Link Bundling

When a HID has multiple active addresses, it is feasible for HIDRA to select a different address for each outgoing datagram; this feature effectively achieves multipath link-bundling. We explored the performance impacts of this approach by altering HIDRA to iterate through the set of active addresses for a HID in a round-robin fashion, and then conducted a simple experiment by starting a throughput test over

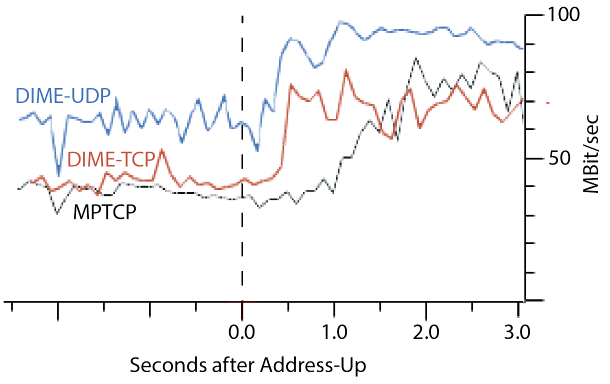


Figure 5.10: Multipath Throughput

one interface and bringing up the second interface.

Figure 5.10 shows the aggregate *throughput* (not goodput) of both links, collected over the course of the interface addition. The results clearly show that (1) DIME responds much faster to address-up events, as expected, and (2) DIME-UDP takes almost full advantage of all available bandwidth. However, the performance of both TCP approaches requires more analysis.

Closer inspection found that, consistent with the results in Sections 5.4.2 and 5.4.3, MPTCP took significantly longer than DIME to establish the new connection. Additionally, the cautious MPTCP slow-start algorithm is clearly visible in Figure 5.10, whereas DIME immediately starts transmitting over both interfaces without hesitation. However, we also found that DIME-TCP's actual application goodput varied wildly, and was consistently below that of MPTCP. We attribute this drop in goodput to TCP's architectural incompatibility with multipath routing, specifically with regards to out-of-order packet delivery.

Abstractly, these results reveal a fundamental challenge facing the Identifier/Locator Split: If transport layer protocols use singular host identifiers that mask locators, then they cannot rely on or optimize for specific network-layer characteristics such as the number of paths. There exist multiple solutions to this challenge, including (1) enacting the above HID address multiplexing on a per-connection basis, (2) integrating MPTCP with DIME to achieve the benefits of both, or (3) adapting TCP's congestion-control algorithm to be more forgiving of multiple routes.

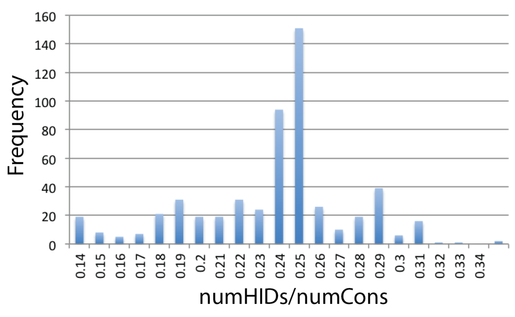


Figure 5.11: HIDs/Cons PDF

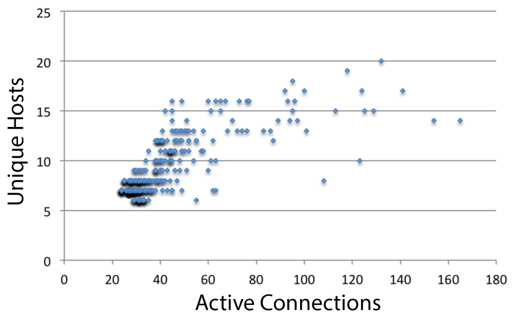


Figure 5.12: Connections vs hosts

The heart of the comparison between DIME and MPTCP across all metrics also raises fundamental and yet-unanswered questions on the nature of routes, end-host addresses, and the relationship between the two with respect to issues such as in-order delivery and congestion-control. However, such a study is well outside of the scope of this paper, and we leave it to future work.

5.4.5 Connections, Hosts, and Scalability

Despite their popularity, a subtle problem of transport-layer mobility solutions is that they must be executed on a per-connection basis. This introduces a remarkable scalability factor, because a mobile host with N active TCP connections must repeat the same migration process N times.

To explore the relationship between hosts and connections and provide a good estimate of N, we wrote a small traffic analyzing tool that logs both the number of active connections and number of unique foreign addresses every five minutes. We then ran this tool across several different network clients as they performed normal network activity. With these results, Figure 5.11 provides a histogram showing that the average value of $\frac{nAddrs}{numCons}$ is roughly $\frac{1}{4}$ (with mean $\mu = 0.242$, and variance $\sigma = 0.04$).

Figure 5.12 provides a scatter-plot of the collected data points themselves. This plot reveals that, while the observed mean numbers are 37.2 connections and 8.5 hosts, the number of hosts grows much more slowly than the number of connections. At the rightmost part of the plot, we find 165 connections across only 14 hosts! From these plots, we conclude that $nConns$ can be approximated as $4 * nAddrs$, but for purposes of scalability, this comparison should really be considered as a lower-bound on $nConns$.

5.4.6 Control Message Analysis

With the results of the above study, we mathematically analyzed the number of control messages sent in response to a network handoff (soft or hard) and the factors that affect this number; Table 5.3 presents a formula for each protocol. This analysis shows that IHMP outperforms its competitors, but *also* reveals three key aspects of how IHMP compares to 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 8-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 based on routing-table information, it is able to accomplish the same example with two exchanges: A1-B1 and A2-B2.

Figure 5.13 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 control signaling than other approaches.

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 5.3: Handoff control messages

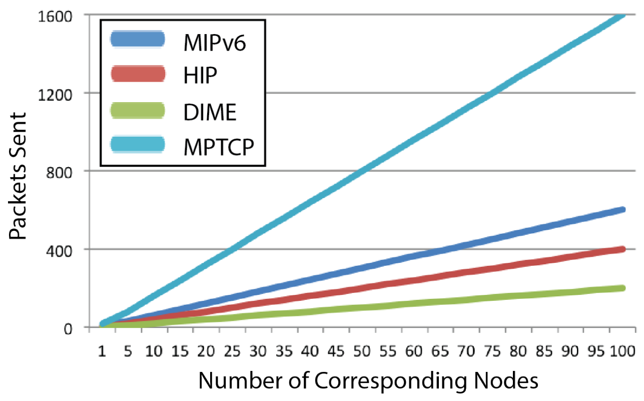


Figure 5.13: Handoff Control Message Scalability

5.4.7 Lines of Code

Table 5.4 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
MPTCP	10,400	0	10,400
HIP	0	28,770	28,770
DIME	200	2,400	2,600

Table 5.4: Lines of Code

5.4.8 Featureset Comparison

Another important consideration for any mobility solution is the *diversity* of mobility cases it supports. Assuming that developers will eventually converge on a single mobility proposal, this proposal must be robust enough to support a wide range of mobility events. Table 5.5 compares the different mobility proposals with respect to many address mobility events that fall outside of our simple testbed scenario, and shows that while different proposals support different features, DIME is clearly the most adaptable and flexible proposal.

Feature	MIPv6	MPTCP	HIP	DIME
IPv4 Support		✓	✓	✓
UDP Support	✓		✓	✓
IPv4/v6 Handover			✓	✓
Private/Link Addrs		✓		✓
Simultaneous Mob.	✓	✓		✓
Preexisting Conns.	✓			✓
NAT Traversal		✓	✓	*
Micro-mobility	✓			*
Multipath		✓		✓
RaspberryPi				✓

*with middlebox support

Table 5.5: Featureset Comparison

5.5 Conclusion

In this chapter, we introduced DIME, a new system for Internet host mobility. DIME combines the HIDRA network stack of Chapter 4 with the Internet Host Mobility Protocol (IHMP); IHMP is an out-of-band end-to-end mobility protocol that updates the address tables of end hosts when they exhibit mobility. We extensively evaluated DIME, and showed that DIME outperforms all existing implemented mobility solutions across a wide range of metrics.

Chapter 6

iDNS: Supporting Information Centric Networking Through the DNS

Chapter 4 introduced the HIDRA network stack in the data-plane, and Chapter 5 shows how a simple control process can be used to augment HIDRA to support network mobility and multihoming. So far, the evaluations in both chapters focus exclusively on the TCP/IP data-plane, and therefore assume the basic (and familiar) case wherein applications use an {ip:port} tuple to identify the service they desire to communicate with. However, this need not be the case! The HIDRA architecture, specifically its introduction of peripheral functions, enables applications to bind a wide range of identifiers to a NID.

In the following two chapters, we show how applications can leverage the HIDRA network stack to support the key goals of Information Centric Networking (ICN) *without* requring a pure ICN architecture or data plane. This chapter focuses on the network application API and content resolution/lookup services, and the following chapter focuses on the in-network content data plane.

6.1 Goals, Assumptions, and Architecture

The key motivator of our approach lies in the observation that the goals of information-centricity lie in the set of benefits attained, not in the methods by which they are attained. Therefore, we seek to distinguish between the common set of *benefits* claimed by ICN proposals and the *characteristics* of these proposals [FLT+13, ADI+12].

Specifically, we enumerate the benefits of ICN proposals as:

- Persistent and unique naming of data.
- Efficient content-distribution.
- Secure content provenance and authentication.
- Better support for network mobility, disruption, and multihoming.

Correspondingly, we enumerate the shared *characteristics* of ICN proposals as:

- Routing based on content names.
- Divorcing content names from location.
- Ubiquitous content-caching at intermediate routers.
- Nearest-replica-routing.

Distinguishing between the common *benefits* of ICN proposals and their common *characteristics* is crucial to our work, because the architecture we propose is a departure from prior ICN proposals - specifically, whereas almost every ICN architecture seeks to replace the entire HTTP/TCP/IP stack entirely, our goal is to preserve the existing network stack in the data plane while still achieving the benefits of ICN.

6.1.1 The Information-Centric Name Resolver

The *Information-Centric Name Resolver* (ICNR) is a peripheral function that maps a content name to a NID that refers to the content's location in the network, as referenced in Section 4.2.2. In this context, we use location to mean a set of one or more open identifiers, specifically {ip:port} tuples. After locating the content object, the ICNR binds all these tuples to a hidden identifier and return this identifier to the application.

Just as there exist several different approaches and proposals for information centric networking, there can exist several different ICNRs. Fortunately, the HIDRA stack is designed to accommodate such diversity and flexibility: since peripheral

Decision	Sample Values
ICNR Input (Step One)	CCN Self-Cert HTTP
ICNR Resolution (Step Two)	DNS Hash Table ICN Routing
Data Plane (Step 3)	HTTP FTP BitTorrent Pure ICN
NID Value (Step 4)	File Contents Exposed Protocol Header

Table 6.1: Example ICNR Variables

functions exist as userspace libraries, we anticipate multiple ICNRs to coexist simultaneously and applications to simply link the appropriate library and make function calls accordingly. Table 6.1 provides a non-exhaustive set of possible choices when designing an ICNR, loosely grouped into (1) the name format accepted by the ICNR, (2) the process used by the ICNR to locate/resolve an NDO, (3) the data plane used to transmit the ICNR, and (4) how the NID is to be read by the application.

In this thesis, we describe a single, specific ICNR that (1) uses a NDN-style name scheme, (2) uses an adapted form of DNS in the location/resolution plane, (3) uses HTTP/TCP/IP (with HIDRA) in the data plane, and (4) exposes the entire HTTP object response to the requesting application for parsing. These decisions stem primarily from ease of development and evaluation in our prototype - aside from the simplicity of deployment, we make no claims as to the value or benefit of these choices over other ones, and invite future work that explores alternate ICNR architectures. We discuss our alterations to the DNS lookup process in the remainder of this chapter, starting with Section 6.2, and our alterations to the HTTP data plane in Chapter 7.

The architecture and data flow of our specific ICNR is illustrated in Figure 6.1. First, the application provides a request for a NDN-style name (e.g. `/ucsc/spencer/thesis.pdf`) to the ICNR as input (Step 1). In Step 2, the ICNR uses the DNS to map this name to a set of n IP addresses that locate different hosts with a copy of the content object. In Step 3, the ICNR creates n separate HID entries (one for each returned IP address), creates n {hid:tcp80} tuples (one for each created HID), and then binds all n tuples to a single NID, which it returns to the application (Step 4). Finally, the application opens a socket to the NID, sends a HTTP request, and processes the response (Step 5).

This two-step approach is a radical departure from existing ICN architectures (e.g. [KCC+07, JST+09, FNTP12, ADM+08]) that propose a single request-response (or publish-subscribe) API. However, this design decision is not made idly: the two-step approach effectively splits the *identifiers* requested by the application (i.e. the NDO names provided to the ICNR in Step 1) from the *locators* used in the data plane (i.e. the NID in Step 3 and its corresponding open identifier bindings). This split is crucial, because only by moving NDO resolution and binding into a separate step can we enable the existing HTTP/TCP/IP stack to be preserved in the data plane.

6.1.2 The DNS Lookup and HTTP Data Plane

Our approach effectively uses DNS to *locate* content objects and HTTP to *retrieve* them. Thus, when we examine this system through the lens of ICN architecture and protocol design, we can say that it uses DNS for the *routing* or *control* plane and HTTP for the *data* plane. We discuss modifications to and design of the data-plane in this section, and discuss the control plane in Section 6.2.

We use HTTP in the data plane because HTTP is the de facto protocol for Internet content delivery today. In addition to the robust ecosystem of browsers and servers deployed at end-to-end systems, we note that HTTP *already* supports transparent caching at intermediate entities. Additionally, HTTP exists entirely as a data plane protocol. This means that aside from using the DNS to resolve a *hostname* to the address of a publisher's servers, HTTP employs no content location or lookup services: HTTP clients simply send content requests, which publishers answer affirmatively (with the content) or negatively (with an error code).

This combination of factors leads us to conclude that HTTP is the most information-

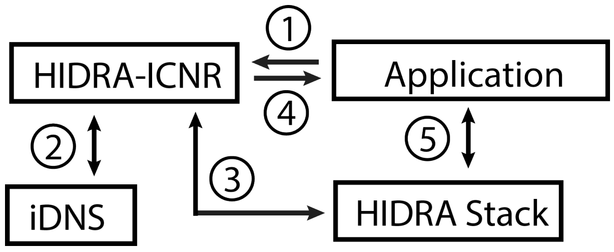


Figure 6.1: The Information-Centric Name Resolver

centric content delivery protocol today, and the best candidate to be adapted towards an information-centric TCP/IP architecture. Specifically, to move from HTTP towards an information centric data plane, only *two* things are needed. First, a content location or lookup service must be deployed at per-object granularity, such that (1) different web objects under the same domain have different content lookup records and (2) non-publisher network nodes can indicate that they have a copy of the object. Second, HTTP must be altered in a manner that supports *transparent* or *opportunistic* caching while still maintaining object security and client privacy. We discuss the first task, that of a lookup service, in Section 6.2. We discuss the second task, that of opportunistic caching in the data plane, in Chapter 7.

6.1.3 Layered Mobility Signaling

The multiple layers of hidden identifiers first illustrated in Figure 4.2 provide many places where open identifier multiplexing can occur in order to mask network disruptions, mobility, and multihoming. Having multiple locations to enact such multiplexing is by design, and illustrates the strength of HIDRA as a modular architecture: rather than concentrating mobility at a single layer, we use identifier multiplexing at different layers to combat different forms of mobility!

Assuming a mobility signaling and mitigating technique such as DIME is employed, the creation of these HIDs is all that is necessary to ensure support for *host mobility.* Each HID will resolve itself to the host's entire set of IP addresses and use end-to-end updates to preserve connectivity across network address changes automatically.

Correspondingly, the set of tuples bound to a single NID supports *content mo-*

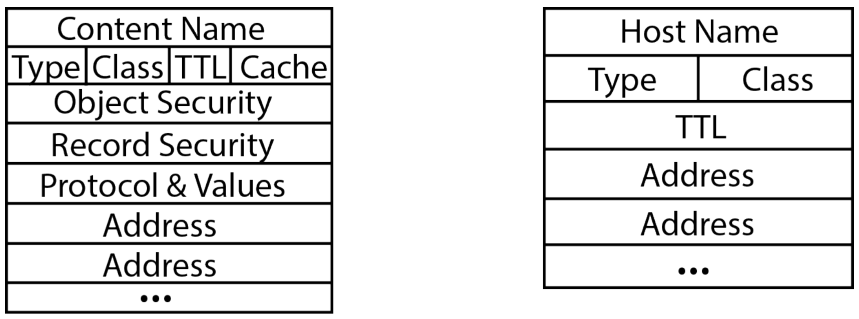


Figure 6.2: Content Record vs Host Record

bility. If a piece of content is located at or replicated across multiple hosts, each content location will have its own tuple bound to the NID. As such, if a specific host becomes unreachable, or the content request fails (regardless of reason), the content request can be multiplexed at the NID and sent to a different host.

6.2 iDNS and the Content Record

Our approach to content location, which we call the *Information-Centric DNS* (iDNS), extends the DNS to denote *content* in addition to *hosts*, and adapts content delivery protocols to reflect this change. While such a shift clearly achieves location-independent content naming, we argue that this shift *also* achieves the other ICN benefits detailed in Section 6.1.

At the core of iDNS is a *Content Record* (CR), which is a new type of DNS resource record that refers to a particular NDO or name prefix. Clients desiring an NDO or name prefix must first resolve the corresponding CR through the DNS, which contains the address of one or more servers hosting the content, along with associated metadata necessary to fetch the content and verify its authenticity.

The Content Record format is illustrated in Figure 6.2 and contains, in addition to the standard DNS resource record fields, a field stating whether the content can be cached, fields for object and record security, a field identifying the content delivery protocol and any protocol-specific values, and a list of one or more addresses where the content can be found. The addresses are included in the response as individual DNS A{AAA} Records. Note that the addresses included are not necessarily those of the publisher or origin, but could potentially be a CDN node, alternate mirrors, or even a nearby cache.

6.2.1 Object and Record Security

The object security field in a CR can take several forms. One example is a hash value calculated over the content. Another is the public-key of the publisher, used by the client to verify a signature provided with the content object. The CR object security field enables the content *record* to secure the content *object*. However, for such a scheme to work, the CR itself must be secured. Given that the CR is just another type of DNS record, the CR may be secured through any one of several existing security approaches proposed to date, such as DNSSEC [WB13].

The process of *creating* a new CR must also be secured, and potentially on a much finer-grain basis than the DNS is today. For example, only Spencer should be allowed to publish CRs under the prefix `/ucsc/spencer`, and only J.J. under the prefix `/ucsc/jj`. This fine-grained security can be accomplished through any of the current access-control techniques used by content servers supporting multiple publishers today, such as HTTP and FTP servers. These servers provide each registered user with their own directory, typically protected by a username and password, which corresponds to a particular prefix or subtree. Given that a particular DNS zone manages the records under the zone, different domains may handle security, registration, and scalability differently without heavily impacting the DNS itself.

6.2.2 Address Record Selection

When a client successfully resolves an iDNS query for an NDO, it receives the CR and one or more address records. In the event that the client receives several address records, it must assume that the records have been ranked by the DNS for locality, availability, or some other metric, and thus should request the NDO from the first address first. Policies may arise and be standardized for address record ranking and ordering, similar to the rules specified in [Dra03] for host IP address selection. However, the logic and considerations involved in such a ranking process is a complex discussion well outside the scope of the iDNS design in this paper.

A crucial part of ICN is directing clients to nearby copies of NDOs. For iDNS to support this functionality, the address of a local content server must be included in the address set, and the address set must be properly ranked to reflect this locality by the time the response reaches the client. Thus, in iDNS we allow any node along the DNS response path to add address records or reorder the records in the set, with the understanding that DNS servers closer to the client have a better understanding of the client's environment.

Though there could be several DNS servers along the return path, in practice there are typically only two: the authoritative DNS server for the record, and the local DNS server for the client. Thus, in iDNS we anticipate the same relationship, and expect that the local DNS server will be largely responsible for directing clients to nearby caches. This approach has an added benefit of fine-grained localization, because the local DNS server sees the address of the client itself. In contrast, the authoritative DNS server sees only the address of the local DNS server, and multiple studies [MCD+02, STA01] have shown that this address is only useful for coarse-grained localization, thereby limiting the effectiveness of CDNs powered by DNS redirection.

6.3 Content Replication

Depending on the address provided to the client in the CR, the client may request the original NDO from the publisher or a *replica* from a nearby cache. We divide content replicas into two forms, long-lived and short-lived, with the difference being that long-lived replicas can generally be relied upon to provide the content, whereas short-lived replicas make no such guarantees. This split is designed to represent the logistical and important difference between hosting entities volunteering to *mirror* content and *caching* entities that work opportunistically. Caches typically provide "best-effort" reliability, given that the requested content may be available, may have been evicted, or may never have been cached before.

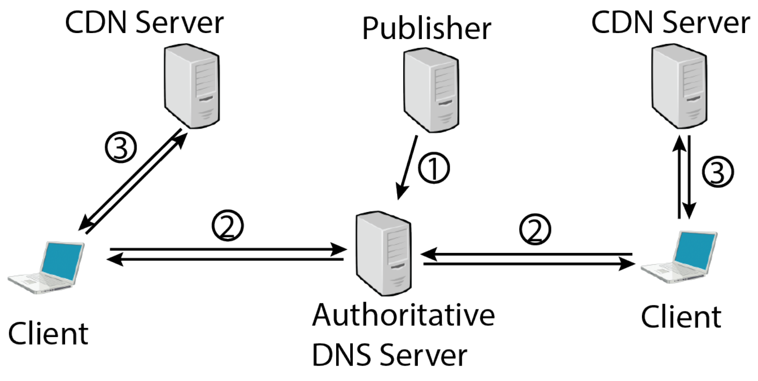


Figure 6.3: Dynamic Record Generation

6.3.1 Long-Lived Content Replication

A publisher may add servers, use mirroring sites, or deploy a CDN to replicate content. In contrast to the ad-hoc methods employed by content delivery protocols today, iDNS provides integrated support for such long-lived replication of content: the publisher simply contacts the authoritative DNS server for the CR and adds an address record referring to the new server hosting that content.

The authoritative DNS server for the CR may order the addresses in a certain way, or only return a subset of the addresses, based on the address of the local DNS server issuing the request. This process is illustrated in Figure 6.3, where the publisher registers the CR (step one), and then two clients using different local DNS servers query the same authoritative DNS server (step two) and receive different address-set orderings. Accordingly, they then request the same NDO from two different content servers (step three).

Distinguishing between *publishing* content (creating new CRs) and mirroring or serving content is important from a security standpoint. Only the owner of a prefix should be allowed to create a new CR under that prefix. However, this same restriction need not apply to parties wishing to mirror or re-host a piece of content. Content mirrors often arise out of immediate necessity [dig], and sometimes the content publisher is either unaware, cannot be contacted, or does not have the necessary resources to scale up content delivery. Thus, other parties may be allowed to append their address to an existing CR without the explicit permission of the

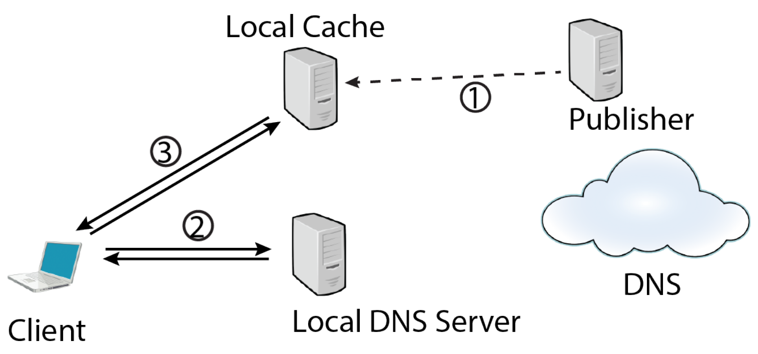


Figure 6.4: Local Record Generation

publisher. However, as long as these parties are not allowed to change the metadata in the CR, including the object security field, clients can easily identify malicious or illegitimate content. Such a restriction can be enacted through the access-control policies mentioned in Section 6.2.1.

6.3.2 Content Caching

Caching is an equally important part of scalable content distribution. Any DNS server on the return path may be aware of a nearby content-cache, and can direct the client to this cache by simply adding the address of the cache to the address set. This process has the potential to be most effective when performed at the local DNS server, since the local DNS server knows the exact network address of the client itself. Figure 6.4 illustrates this process, where after a content cache receives an NDO (step one), the local DNS server directs the client (step two) to request the NDO from this cache (step three).

6.4 Comparing iDNS to Prior ICN Proposals

Having provided a technical overview of iDNS, we return to the previously stated goals and benefits of ICN with the intent of showing that we achieve all of the benefits of prior ICN proposals.

6.4.1 Location-Independent Persistent Naming

iDNS ensures that content names are persistent and unique through the hierarchical nature of the DNS. It also decouples names from locations by separating the CR from its set of addresses, and maintains a namespace that is not fragmented, even when content is moved or mirrored across different content servers.

There exists ongoing debate [GKR+11, BC12] on whether content names should be drawn from hierarchical or flat name spaces. Interestingly, flat-name proposals necessitate a peripheral name resolution service (NRS) to translate between user-readable names and routable ones, and various works within this space argue whether this NRS itself should be flat or hierarchical.

In this debate, iDNS does not advocate a particular approach over another: the CR provides a natural point of convergence for either approach! Though the DNS itself is hierarchical, architecturally flat name resolution protocols exist [KFV+12, RS04, VBZ+12] and other protocols can be designed as necessary: they must simply map a name to a CR. Powered by the modular design of HIDRA, the ICNR provides a location to easily adapt and extend information-centric name resolution however necessary. Moreover, the DNS itself can be easily adapted to a flat naming scheme, as is proposed by NetInf [ADM+08].

6.4.2 Efficient Content Distribution

The primary motivation for ICN is to relieve network congestion and improve content distribution through a combination of caching and nearest-replica-routing. The iDNS content location process accomplishes this goal by enabling servers along the DNS response path to change the cached address-set via the guidelines in Section 6.2.2, and supporting already-existing opportunistic Web caching techniques in the HTTP data-plane.

This brings us to another ongoing debate in the ICN community [FLT+13, GSK+11, XVT+12, CHPP12, DBGLA14] regarding the effectiveness of different caching policies; this debate typically compares *ubiquitous caching* in the core of the network to *edge caching*. Again, without claiming either side in such a debate, iDNS can enable any caching policy, depending only on the topology of intermediate DNS servers appending cache addresses. This provides a systematic answer to the debate, because network operators will only place additional DNS servers at locations that they deem most effective - therefore, the caching topology will naturally evolve and converge on the most ideal model.

6.4.3 Object Level Security Model

An important ICN design primitive is the concept that content can come from any location in the network. Thus, the traditional security model, which focuses on securing and authenticating *hosts*, must be changed to authenticate and secure *content* instead. iDNS preserves the concept that content may come from anywhere, and accomplishes object-level security through the security field in the CR.

Compared to other ICN proposals, an advantage of iDNS is that its integrity model depends *only* on the DNS. As long as the base CR is secured, via DNSSEC or some other protocol, then the client can easily verify the integrity of the received content object. Furthermore, iDNS does not require any intermediate routers to verify the authenticity of the content. This technique avoids an open problem in the ICN community, where many questions exist regarding the trust and feasibility of a universal PKI (or other such security protocol) deployed at intermediate routers, as well as the feasibility and scalability of performing content verification at each router.

Finally, we note that security *also* refers to other concerns, most importantly client privacy. Preserving the privacy of content objects and consumer requests is an open and major problem in ICN, with recent works [GTW16] arguing that strong privacy guarantees are fundamentally untenable in ICN. In contrast, when HTTP is used in the data-plane, there exist several techniques and approaches that can be used to mitigate this problem in different forms; we discuss data-plane security further in Chapter 7.

6.4.4 Mobility And Disruption

iDNS provides natural support for many different forms of mobility. Client localization can be enacted by leveraging DHCP to provide clients with the address of a nearby local DNS server when they join a new network. Content mobility can be supported by (1) updating the address set in a CR to reflect new replicas and (2) using NID multiplexing in the data plane to persist a single HTTP session across host failures or disconnections. Finally, by means of the HIDRA architecture, seamless *host* mobility can be supported via techniques such as DIME.

6.4.5 Differences With Prior ICN Work

Architecturally, iDNS differs from other ICN proposals in two key fashions: First and foremost, it preserves the HTTP/TCP/IP data plane by augmenting it with DNS resolution. Second, it uses two request-response pairs, one to locate the content and the other to fetch it. This approach contrasts with other ICN proposals, which typically employ a single request-response pairing. Conceptually, this separates the act of *locating* content from the act of *distributing* it, and this split enables two separate topologies to coexist: one for content-location and the other for content-distribution. This design is a key strength of iDNS, because it effectively supports "near-replica routing" without relying on large content tables or a content-routing protocol. DNS names are routed swiftly without any localization or fragmentation, and then the content-request itself is routed over IP.

This split has another important ramification, in that it enables both steps in the system (content location and distribution) to evolve independently of each other, bridged only by the format of the CR. Hence, iDNS can support multiple different approaches to naming and caching, as well as a large suite of alternative content delivery protocols, including FTP, BitTorrent, and Gnutella, among others.

6.5 Analysis of Scalability

Supporting NDO resolution through the DNS increases the number of records in the DNS by several orders of magnitude, roughly from 10^6 to at least 10^9 [PV11, big]. Though the DNS is known to be a highly scalable distributed system, such a significant increase in scale merits further examination. In particular, we examine the scalability of two key parts of the system: the authoritative DNS servers in charge of storing and serving the records; and the local DNS servers in charge of forwarding queries, caching entries, and returning records to clients.

6.5.1 Scalability of the Authoritative DNS

By increasing the number of records served by the DNS, we implicitly increase the amount of (a) storage and (b) processing power necessary to serve these records. Additionally, if we increase the average name length (a likely assumption), we potentially incur additional DNS referrals.

Storage and Processing Power

Increasing the number of records in the DNS correspondingly increases the work necessary to store and serve these records. However, it can be qualitatively argued that a comparable amount of work is already performed today by HTTP servers. Given that today's DNS resolves nothing more than a hostname, an HTTP server must manage an entire directory tree, parse the HTTP path accordingly, and return the necessary piece of content. In contrast, DNS servers must only return a corresponding CR, not the content object itself. Even today, the performance of TLD servers (e.g., `com` or `org`) shows that a particular DNS zone *can* support thousands of entries.

Fortunately, the DNS is a well-designed, hierarchically distributed system. This design ensures that if an organization struggles to serve or update their CRs, this inadequacy is contained to the CRs of this organization and does not slow down or create problems elsewhere in the DNS. Thus, there exists a powerful and natural motive for an organization to successfully manage the publication of their content and provision adequate resources to do so. Additionally, the impact and consequences of negligence or failure to provision resources accordingly is limited to the offending organization(s) and does not negatively impact CR resolution under alternate namespaces.

Latency and Referrals

DNS requests start at the root and descend the hierarchy as necessary. For example, with no cached information, DNS resolution for `bsoe.ucsc.edu` consists of three requests: the first to the root name-server, the second to the authoritative server for `edu`, and the last to the authoritative server for `ucsc.edu`. Thus, as names contain more components, they necessarily result in more requests and referrals.

This design means that the behavior of DNS, in particular referrals, depends on the structure of the content name: the same set of records may result in different behavior, depending on how their names are structured. Accordingly, to provide a meaningful analysis, we had to make assumptions about the distribution and structure of content names. In particular, we assume that the structure of content names in a DNS-based system mirrors the structure of HTTP names used today: for example, the URL `bsoe.ucsc.edu/index.html` would correspond to four iDNS zones, with the zone `bsoe` being in charge of the CR for `index.html`. Such an assumption is safe and useful: safe because HTTP does not mandate the format of the path component, and useful because it enables us to draw conclusions from existing HTTP names and traffic.

Building on this assumption, we analyzed a large set of HTTP GETs[1]. In our analysis, we stripped out the hostname and then examined the rest of the HTTP path for the number of components. For example, a GET for `bsoe.ucsc.edu/index.html` would have a value of 1, whereas `bsoe.ucsc.edu/videos/index.html` a value of 2. Our results are shown as a histogram in Figure 6.5, with a mean value of 3.9 and standard deviation of 2.89.

[1]One day (2012-11-01 00:00∼23:59) of HTTP traffic initiated by hosts at POSTECH University in South Korea, approximately 25 million requests

Notably, prior analysis of DNS traffic has shown that DNS requests and referrals are largely mitigated through local DNS caching. Jung et al [JSBM02] observe that the average DNS query results in 1.2 *referrals* and a latency of approximately 60ms, despite the fact that the average DNS name has 3.3 *components*. These results are encouraging, because they illustrate the effectiveness of caching in improving DNS performance.

Based on the above results, we believe that caching and other optimizations used for host-name resolution with the DNS will be equally successful when extended to content objects. When deployed at a large scale, we therefore expect the average name to consist of ~ 6.8 components, and to result in ~ 2.4 referrals and an average latency of ~ 100ms, all of which are acceptable values.

6.5.2 Scalability at the Local DNS Server

iDNS increases the work required by the local DNS server, because it must direct clients to nearby content-caches. In its simplest form, redirection is accomplished by appending an address to the address set of each DNS response. This operation, which must happen for each request/response pair, constitutes less work than a transparent cache carries out today when it inspects HTTP headers. More complex schemes for cache load balancing may evolve, but such schemes represent a fundamental tradeoff between additional complexity at the local DNS server or decreased efficiency at the content cache. This tradeoff is important to highlight, because such a tradeoff can only be examined and optimized for a particular local topology and set of hardware, yet we show that iDNS provides support for such optimizations.

Given that local DNS servers often cache records to improve DNS performance, increasing the number of DNS records can have an adverse affect on the local cache. However, multiple studies [JSBM02, CK03] indicate that the DNS cache size is not a limiting factor on performance, because the distribution of DNS objects is Zipf, and the individual record objects are quite small. In fact, DNS objects are so small that the common DNS caching utility `dnscache` provides a default cache size of 1MB and a *maximum* cache size of 16MB! Thus, there is ample room for DNS caching to

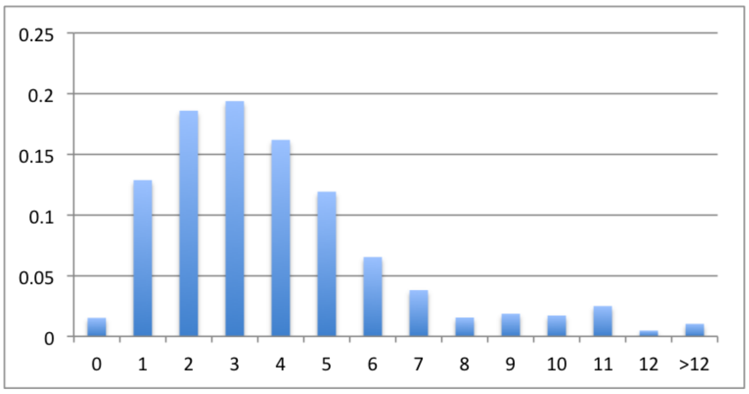


Figure 6.5: Histogram of HTTP Path Components

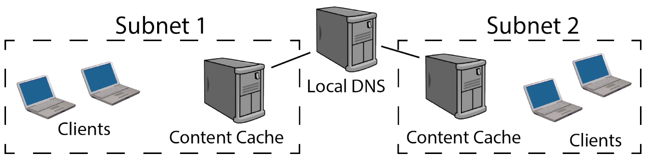


Figure 6.6: Prototype Deployment Topology

expand by several orders of magnitude before an impact on performance is felt.

6.6 Experimental Deployment

To explore a common deployment scenario, we built a prototype iDNS system that employs hierarchical CCN-style naming, edge-caching through local DNS servers, and HTTP for content delivery. We wrote a simple iDNS client, local DNS resolver, and content cache in Java, and deployed the code (approximately 2000 lines, of which only 200 are unique to iDNS) across three servers and four clients at PARC configured in the topology shown in Figure 6.6. Both subnets use the same local DNS server, which directs clients to their closest cache based on their address; the primary difference between the two subnets is that the cache in Subnet 1 is directly along the network-path from the clients to the Internet, whereas this is not the case for Subnet 2. To avoid changing the authoritative DNS server, we elected to encode CRs as a TXT record starting with `"CR:"`.

6.6.1 Name Format Translation

We start with a hierarchical content name (e.g., `/ucsc/ccrg/papers/idns.pdf`), which is translated to a DNS-resolvable name through a simple algorithm: First, reverse the order of all names broken by the `/` character to create the string `idns.pdf/papers/ccrg/ucsc/`. Next, swap each `/` character for a `.`, and each `.` for a `/`, to create the valid[2] DNS name `idns/pdf.papers.ccrg.ucsc`. This simple translation is one-to-one and reversible, which allows the DNS name to later be reconstructed into the original hierarchical content name.

To support the HTTP data-plane, we define a *hostname length number* (HLN) to be included with the CR. The HLN is used to translate the content name from DNS to HTTP; this is necessary, given that HTTP URLs contain two hierarchical components, the hostname and the path. Thus, the HLN is needed to denote the number of components in the hostname, with the assumption that the remainder of the name is the content path. For example, when $HLN = 2$, the DNS name `idns/pdf.papers.bsoe.ucsc.edu` translates to `ucsc.edu/bsoe/papers/idns.pdf`, whereas $HLN = 3$ would create `bsoe.ucsc.edu/papers/idns.pdf`. Once translated, the client then issues an HTTP GET request for the constructed URL.

6.6.2 Latency Results

We hosted a 456KB file on a server at UCSC, and created a CR naming it as `edu/ucsc/soe/ccrg/idns.pdf`. We then had each client in our test topology request the file 10 times, using three different caching schemes: first without caching, second only using transparent caching along the network path, and third using iDNS cache location to explicitly address the same cache.

The first row of Table 6.2 shows the average latency (μ) and variance (σ) of the HTTP transfer, as perceived by the end client. As expected, these results show that a cache-hit reduces latency as compared to fetching the object from the origin; however, they *also* show that there is minimal performance difference between trans-

[2]DNS explicitly prohibits use of the "/" character in *hostnames*, but allows it in other record types, such as TXT or our CR.

Cache Policy	None	Transparent	iDNS
Client Latency	$\mu = 45ms$ $\sigma = 5.03ms$	$\mu = 26ms$ $\sigma = 2.71ms$	$\mu = 29ms$ $\sigma = 3.45ms$
Cache Latency	N/A	$\mu = 52ms$ $\sigma = 4.47ms$	$\mu = 61ms/32ms$ $\sigma = 4.67ms/2.94ms$

Table 6.2: Results

parent in-line caching and our method, which directly addresses the cache itself and includes the origin addresses as an HTTP header option. This result is important to our design because it enables clients to take advantage of caches existing outside of the direct network-layer path to the server.

The second row contains the average time needed to populate the cache itself the first time the file is requested. When requesting a file from the origin, iDNS exhibits slightly more overhead compared to transparent caching (61ms to 52ms); this overhead is the natural result of coordinating two separate HTTP requests as opposed to simply sniffing and copying data. However, the second entry under iDNS (32ms) shows an interesting observed behavior: the first time an iDNS cache requests the file, it must be served from the *origin* server at UCSC, yet when the second cache requests the same file, it can locate and request it directly from the first iDNS cache. This behavior results in lower latency as well as distributing the load off the origin server.

6.7 Conclusion

In this chapter, we introduced, explained, and evaluated iDNS, a novel approach to Information Centric Networking. iDNS shows how the hidden identifier architecture in Chapter 4 can easily be extended to semantically bind named data objects instead of hostnames. We introduced the concept of an Information Centric Name Resolver (ICNR), showed how different ICNRs can support different information centric paradigms, and showed how a specific ICNR can be used to leverage the existing DNS and HTTP protocols.

To support our specific ICNR, we proposed extending the DNS to resolve content names as well as hostnames; we call this system iDNS. We show that iDNS maintains compatibility with existing approaches to routing and content-delivery, and requires only minuscule changes to end clients. This compatibility means that iDNS can be deployed *today*, yet can still be extended to support future developments (e.g., content routing and content security) in other ICN architectures as they mature. Our analysis of DNS and HTTP shows that iDNS can feasibly be deployed at Internet scale, and our prototype deployment shows that iDNS achieves the benefits of ICN without incurring significant processing or control overhead.

Chapter 7

GroupSec: A New Security Model for the Web

Chapters 3 to 5 of this thesis combine to paint a compelling picture of a future Internet, wherein identity layers are cleanly separated and mobility within a layer does not interfere with other layers. Building on this architectural framework, Chapter 6 showed how the goals of ICN can be attained over the present-day HTTP/TCP/IP. This design is exceedingly important, because the initial ICN proposals [KCC+07, JST+09, FNTP12, ADM+08] correctly observe that the single dominant purpose and function of Internet communication today is content delivery.

Chapter 6 showed that the goals of ICN can be attained over HTTP/TCP/IP provided that (1) a fine-grained and lightweight content lookup service (i.e. iDNS) is deployed and (2) HTTP itself is altered to support the content-centric security model. Digging deeper into Point (2), we argue that the key challenge of adapting HTTP today to a purely information-centric protocol lies in two facts. First, a content-centric security model must support transparent, ubiquitous, opportunistic content caching, wherein individual content objects can come from anywhere in the network. Second, secure HTTP (HTTPS) uses a *session-based* security model wherein clients authenticate *endpoints* (i.e. a publisher's server) and then rely on end-to-end encryption to ensure content integrity and privacy.

HTTPS is currently used for slightly over half of all traffic flows, [NFL+14] and its adoption is on the rise as a response to growing concerns of Web privacy and security [htta]. Unfortunately, the session-based security model clearly interferes

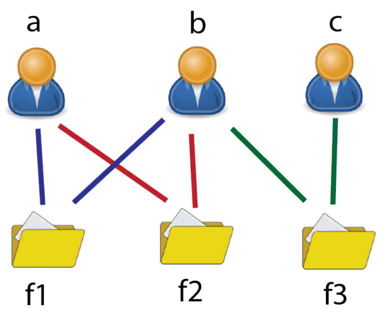


Figure 7.1: Content Group Membership

with all forms of Web caching. It follows that the single most pressing obstacle facing content publication and dissemination on the Web today is the question of how to integrate Web security and privacy with transparent content caching; this chapter focuses on this challenge.

7.1 Content Group Security

The key innovation of GroupSec is a new security model based on *group membership*. In this model, clients are defined as being in a "content group" together if they are authorized by the publisher to view the same content object. Content groups exist separately for each content object, and may overlap. Figure 7.1 provides a simple example, where users a and b are in two content groups together (groups $f1$ and $f2$), and user b is also in a third content group ($f3$) with user c.

Compared to TLS, GroupSec relaxes the security restraints on content groups in two key ways. First, nodes within the same content group are allowed to infer each other's membership with respect to that particular group (i.e. if a node is authorized to view a content object, it can also deduce when other clients are viewing the same content object). Second, nodes *outside* the group may see that clients are in a unique group together, but cannot deduce anything about the nature of the group.

Continuing the example in Figure 7.1, user a can see that user b is able to access files $f1$ and $f2$. Likewise, user c can see that users a and b are in two distinct content groups together, but cannot access either file or its filename.

7.1.1 Asymmetric Privacy Model

The group membership security model is carefully designed to meet the concerns of publishers distributing a file to multiple clients. Specifically, GroupSec is designed around the observation that in this scenario, publishers and clients have asymmetric privacy concerns! From the perspective of content publishers, privacy refers to the nature of the file itself (i.e. its name and contents) that the publisher is serving. Conversely, from the perspective of clients, privacy refers to the nature of the file, but *also* the fact that the client requested that specific file.

Subdividing privacy in this manner is a crucial part of the GroupSec design, because it enables minimal leaking of information while still supporting transparent caching: for a transparent cache to operate, it must know when multiple clients request the same content object. It follows that the core goal of the GroupSec privacy model is to expose this information, and *only* this information, while still protecting the name and content of the cached files *even from the caches themselves!*

Publisher Privacy

From the specific perspective of content publishers, GroupSec achieves a level of security equivalent to HTTPS. First, since only clients in a content group are able to decrypt and view the relevant content object, publishers are assured that knowledge of a file's name and contents is restricted only to those clients the publisher has authorized. Second, in GroupSec the identity of the publisher (i.e. its hostname) is known to all the clients in the group, and the IP address of the publisher is known even to nodes outside the content group, but this is *already* the case today for any publisher serving content over HTTPS.

Client Privacy

In contrast to the perspective of publishers, GroupSec clients see a significant *downgrade* in terms of privacy when compared to HTTPS. Whereas HTTPS offers clients assurance that no one but the publisher knows anything at all about their content request, GroupSec relaxes this restriction such that (1) other nodes in a client's content group can see that the client has requested the file, and (2) nodes outside the content group can identify the members of a unique content group by IP address.

This relaxation raises significant privacy concerns, because a client may not necessarily trust every other client in a content group, and may not wish other network nodes to identify that it is requesting the same content as other nodes. Because of these concerns, GroupSec represents itself to clients as a completely *non-private* connection. That is to say, end clients using GroupSec can be assured of the *authenticity* and *integrity* of the content, yet are given no assurances about the *confidentiality* of their request.

Defining a connection as authenticated, yet non-private from the perspective of a single side of the connection is a significant departure from all prior security models. However, such a definition is not only a good fit for the Web, it is also remarkably easy to convey. GroupSec leverages the observation that Web users most often use "secure" to mean "private" [TZY01, KHY12], and simply uses the browser lock icon to identify GroupSec content as "insecure".

7.1.2 HTTP-Centric Security

Instead of layering HTTP traffic on top of a TLS session, GroupSec enacts security within HTTP itself. GroupSec accomplishes this by encrypting the filename and contents separately though an out-of-band process similar to [Tho16], and then transmitting HTTP requests and responses over plaintext. Figure 7.2 illustrates this process and shows which specific fields of the HTTP request and response are encrypted.

This shift is incredibly important: while GroupSec ensures equivalent security to TLS with respect to the Request-URL and Message Body fields, intermediate nodes may view and modify all other HTTP header fields. While this decision can be seen as "relaxing" the traditional security model by exposing more information to intermediate network entities, we argue that this shift actually comes with several important benefits and minimal drawbacks. First, even when TLS is used, an attacker can still infer that two nodes are exchanging HTTP traffic simply by observing the IP addresses, ports, and communication pattern between the client and publisher. Second, we note that HTTP header modification is a common bandwidth-saving technique employed by transparent caches to encourage content reencoding for mobile devices [NFL+14].

7.1.3 Middleboxes and Trust

A remarkable drawback of prior work [NSV+15, SLPR15] is that they explicitly authorize certain middleboxes to view and/or modify content. This design is fundamentally at odds with the observation that middleboxes are not necessarily trustworthy, and in some cases (e.g. ISPs injecting additional advertisements) should be considered malicious. Thus, rather than opening debate or proposing mechanisms to separate "good" middleboxes from "bad" ones, we assume a simpler trust model wherein clients trust content publishers and no one else.

7.2 HTTP-GroupSec

For the GroupSec model to be feasibly deployable in the Internet today, it must be implementable with minimal changes to browsers and servers, and must not depend on alterations to middleboxes themselves.

In applying the abstract security model of group membership to HTTP, we found that four key requirements dictated our protocol design. Formally, a content object must (1) be decryptable by the intended clients, (2) not be decryptable by other nodes, (3) be cacheable by intermediate entities, and (4) fully mask the name of the object.

Our solution, which we call HTTP-GroupSec (HTTP-GS), starts with the assumption that a HTTPS session exists between a client and publisher, and that this session was used to load a preexisting page, which we call the *linking page*. Through the use of two new HTML attributes, `http-gs-key` and `http-gs-salt`, the linking page indicates to the browser that certain elements (either embedded or linked) are HTTP-GS enabled. The browser then retrieves these specific elements over HTTP, decrypts them, and renders them in the page accordingly.

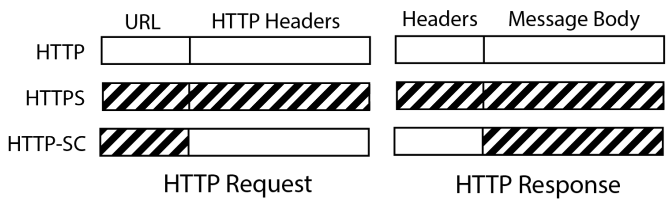


Figure 7.2: HTTP-GS Requests and Responses

7.2.1 Object Encryption and Decryption

Before they are linked or served, HTTP-GS content objects are encrypted with their own public-private keypair. In keeping with current recommended best practices [rsa, cer, sta, ope, BBB+06, BR11, FP] we use a 2048-bit RSA key, but stress that the HTTP-GS design can support any form of asymmetric encryption. To support fine-grained object-level security and enable different content groups to emerge for each individual object, each HTTP-GS content object is encrypted with a separate key. The public key itself is transmitted over HTTPS as a part of the linking page under the `http-gs-key` link attribute; this enables clients to be assured of both the integrity and confidentiality of the key itself. Once encrypted, the HTTP-GS content object itself is simply served as any other HTTP object - the only difference being that the Message Body of the HTTP response is encrypted.

7.2.2 URL Hashing

HTTP URLs are comprised of two parts: the Hostname (e.g. `www.example.com`) and the Path (e.g. `/videos/v1.mpg`). Since both fields are transmitted in cleartext as part of the client's request, they must both be sufficiently masked to ensure privacy. We use two different techniques to encrypt each component separately.

The Path requested by the client is generated by hashing the URL provided by the linking page; we call this value the *name-hash*. In generating this hash, we have two goals: First, nodes that are not members of the content group must not be able to reverse the name-hash to the original URL. Second, to leverage transparent caching, the name-hash must be *consistent*, so that multiple clients requesting the same content object refer to it by the same name-hash.

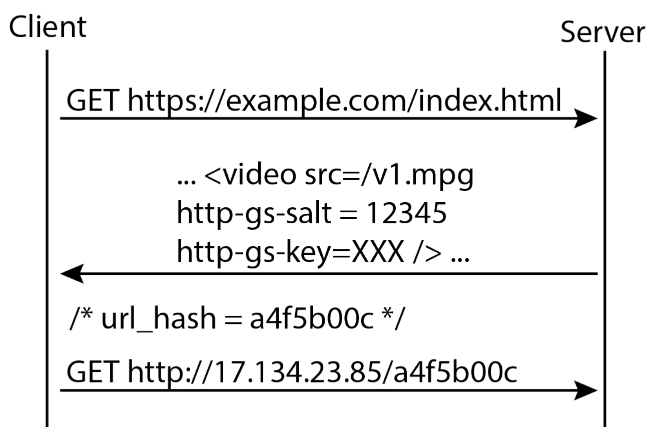

Client
Server
GET https://example.com/index.html
... <video src=/v1.mpg
http-gs-salt = 12345
http-gs-key=XXX /> ...
/* url_hash = a4f5b00c */
GET http://17.134.23.85/a4f5b00c

Figure 7.3: HTTP-GS URL Hashing

The name-hash is generated by including another attribute, `http-gs-salt`, in the HTML of the linking page. This attribute has two values, the salt itself (a randomly-generated number provided by the publisher) and an expiration date.[1] The browser verifies that the salt is within the expiration date, and then creates the name-hash by adding the salt, the key, and the URL together, and then calculating an md5 hash of this value as in Equation 7.1.

$$name\_hash = md5(url + key + salt) \tag{7.1}$$

Hashing the URL in this manner ensures request consistency: since every client receives the same URL, key, and salt from the publisher, every client recreates the same name hash. Additionally, the salt serves to place an expiration date on a HTTP-GS link: even if an eavesdropper manages to possess the content object's key, either by compromising a HTTPS session or by having been authorized to receive this content in the past, just owning the key is *insufficient* to generate a request URL or decrypt a name-hash. This has broad-reaching implications for client privacy, performance optimization, as well as DRM and key revocation.

[1]Choosing a good expiration time, or refresh-rate, is left to the discretion of the publisher, since it dictates a tradeoff between privacy and cache efficiency. We strongly recommend that this value be closely coordinated with the `Cache-Control: expires` header.

7.2.3 Hostname Stripping

HTTP-GS requests effectively remove the Hostname field entirely, and simply name the host by IP address. This decision to replace the hostname with an IP address comes from several motives. First, hashing the hostname separately for each web object, the way the name-hash is generated, risks a cross-domain hash collision at intermediate caches (e.g. `domain1.com/fileX` and `domain2.com/fileY` both hash to the same value). Such a collision poses a serious problem, because it could disrupt client access to the content (i.e. a client requests one file and the cache delivers the other), yet this behavior would be completely undetectable (and therefore uncorrectable) by content publishers.

To prevent such collisions from occurring at transparent caches, each domain must be given a unique and consistent namespace to generate hashes in. This makes hash-collisions within a domain easy to identify (e.g. a publisher can immediately detect if `domain1.com/fileX` and `domain1.com/fileY` map to the same hash), yet this poses an equally important security leak: if a hash used to mask a domain name must be consistent for every object named under the domain, an attacker could discover the Hostname hash by simply visiting any public-facing page in the domain.

By replacing a publisher's hostname with it's IP address, we resolve both problems at once. The consistent value of the IP address removes the threat of hash collisions at transparent caches, and does not expose any new information not already visible in the IP header. In the case of CDNs or other facilities that host multiple domain names at a single physical IP address, potential collisions are just as trivial to identify and correct, and actually serve to further obfuscate the name of the content object requested: a request of the form `ip:content_hash` leaks no information at all if `ip` is the address of a server known to host several different websites.

7.2.4 Transparent Caching

Transmitting requests and responses over plaintext HTTP enables Web caches to consistently identify, store, and serve HTTP-GS content in response to future requests. Additionally, leaving the HTTP headers unprotected allows these same caches to (1) read and act on relevant cache-specific HTTP headers such as `Cache-Control` and (2) add specific headers to outgoing HTTP requests, such as requesting a mobile-specific version of the object if one exists.

7.2.5 Cross-Domain Linking

One of the key motives in our choice to use public-private keypairs, as opposed to symmetric keys, is to support HTTP-GS linking across domains. By adding the `http-gs-salt` and `http-gs-key` attributes to a link tag, a page loaded over HTTPS can securely embed or link to elements outside of its domain. This enables integrated support for personalized content or aggregator sites (e.g. Reddit or Google News) and highlights the strength of HTTP-GS. While the initial personalized or aggregated site must be loaded over HTTPS, every subsequent linked or embedded object can be loaded over plaintext and cacheable HTTP! We anticipate this specific use model to account for a large portion of the network benefits of HTTP-GS.

For such a design to work, the `http-gs-salt` attribute must either be (1) set to a sufficiently large value or (2) updated by the publisher every time it changes. However, we anticipate that a simple protocol could support this feature automatically. More importantly, similar to key revocation for clients, this design puts an "expiration date" on cross-linking websites, since a publisher can revoke access by simply denying a linker's request for the current salt.

7.3 Threat Model Analysis

In this section, we examine GroupSec and compare it to HTTPS with respect to a large range of common attack vectors. We primarily consider two attacker models: an attacker that is *not* part of a client's content group, and an attacker that is part of the client's content group (i.e. the attacker possesses the current salt and key). We explicitly do not consider attack vectors that lie orthogonally or out-of-band with respect to HTTP-GS and HTTPS (e.g. an attacker gaining physical access to a server or breaking public-key cryptography) since such vectors are out of the scope of this paper.

7.3.1 Unauthorized File Access

The primary privacy concern of publishers is that unauthorized clients will gain access to their content. However, since content is encrypted with the publisher's private key prior to distribution, and the public key is only distributed over HTTPS, an attacker will be unable to decrypt HTTP-GS content. To claim otherwise is to say that either (1) the attacker was able to obtain the key by hijacking an HTTPS session or (2) the attacker was able to break public key encryption.

7.3.2 Client Requests

Other attackers, such as surveillance organizations, may simply wish to learn that a client requested a specific file. In these cases, the attacker's ability to do so hinges on membership in the content group.

In cases where the attacker is not a member of the content group, we assume that the attacker is not in possession of the filename, key, or salt. It follows that without at least two of these values, the process of reversing a name hash to a {filename, salt, key} tuple is fundamentally impossible, even if the hash function used is reversed! This is because even if the attacker is able to break the hash function, the attacker will simply obtain the sum of these three variables, without any further information as to which value is which.

In those cases where the attacker *is* a member of the content group, the process of identifying the file requested by the client is trivial: by simply recreating the current name hash, an on-path attacker can immediately detect whenever the file in question is requested by a client. However, this case is *explicitly* allowed by the security model, and thus is not a violation.

More abstractly, GroupSec addresses this threat by portraying GroupSec content to clients as non-private: [LR07, RLC05, ERB03] have found that Web users alter their behavior based on privacy indicators and perceived privacy. This ties in to our anticipated use model, that HTTP-GS will be primarily used by publishers to "upgrade" the security of relatively non-private content (e.g. movies, news articles, etc.) in a way that protects the publisher's interests (i.e. DRM and client authentication). We stress that HTTP-GS is a poor fit and not intended for private or sensitive communication (e.g. email), both in that it does not guarantee the same client privacy as TLS and that this content is most likely not cacheable.

7.3.3 Content Spoofing

In a content spoofing attack, on-path attackers respond to intercepted content requests with a fake piece of content. However, for such an attack to work on HTTP-GS content, the fake content must have been encrypted with the correct key - otherwise, client-side deencryption will fail. Since HTTP-GS uses asymmetric keys, and the publisher's *private* key is never even transmitted over the network, this attack cannot succeed unless the attacker either breaks public-key encryption or obtains the publisher's private key through some form of offline attack.

7.3.4 Cache Poisoning

A cache poisoning attack is similar to a content spoofing attack, except that the goal of the attacker is simply to disrupt client access to content by populating a cache with incorrect or false objects. Even if clients detect that the content is spoofed, cache poisoning can still occur because if a cache stores this incorrect object, it will respond to all subsequent requests with the same incorrect content.

HTTP-GS protects against cache poisoning attacks that seek to disrupt access to a specific content object by ensuring that attackers cannot generate the name-hash for a specific content object; without access to the name-hash, attackers cannot pick out the specific piece of content to attack. Wide-range cache poisoning attacks (wherein an on-path attacker replies to every HTTP request with a fake content

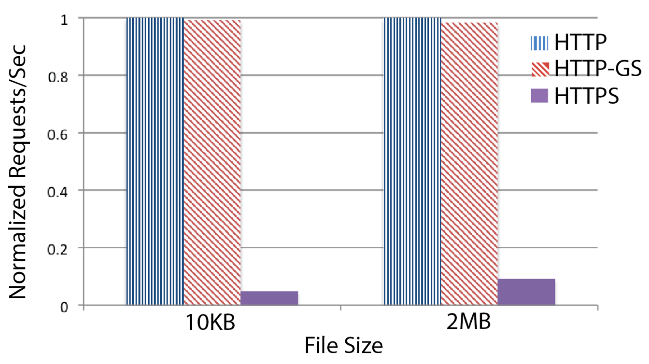


Figure 7.4: Server Load

object) are still possible, yet unlikely - attackers in such a position can simply execute a DoS attack by silently dropping the HTTP request packets.

7.4 Performance Evaluation

We implemented a GroupSec prototype in Javascript and deployed it on a simple testbed consisting of two laptop computers connected over ethernet: a client running Firefox and a server running Apache. We then used this testbed to evaluate GroupSec performance in two key metrics: *sustainable load* at the content server and *page load latency* at the client. We chose these specific metrics and topology because (1) load and latency are the two metrics most important to Web publishers and (2) they compare GroupSec at its absolute worst (i.e. no transparent caching).

7.4.1 Sustainable Load

The key benefit of GroupSec is that it enables transparent content caching; such caching will clearly serve to decrease the load on a publisher's servers. However, since transparent caches are not ubiquitously deployed, GroupSec must not incur additional load when caches are not on the path. We recorded the sustainable load at the server, measured in requests-per-second, by running an Apache Benchmark test from the client to the server; this test repeatedly requests the same URL over HTTP, HTTPS, or HTTP-GS. Figure 7.4 contains our results for two filesizes: the Apache2 default page (∼10 KB), and a larger file (∼2 MB) chosen to reflect the current average Web object size [httb]. To provide a platform- and filesize-independent comparison metric, we normalized the results by the collected HTTP values (8084 RPS for the 10K file and 1161 RPS for the 2M file).

These results show that the load of serving a HTTP-GS object is roughly equivalent to serving a regular HTTP object; this is unsurprising, given that HTTP-GS objects are transmitted over regular HTTP. More importantly, Figure 7.4 *also* shows that HTTP (and HTTP-GS) both vastly outperform HTTPS, by 20x and 10x, respectively! These results show that HTTP-GS helps to dramatically increase the sustainable load on a publisher's server even when transparent caching is not employed.

7.4.2 Latency

Client-perceived latency is arguably the most important metric for content publishers [Ham]. We explored the effects of HTTP-GS on page load latency by creating a webpage with ten separate images, hosting it on the server, and migrating the images one-by-one from HTTPS to HTTP-GS. Figure 7.5 compares these results to the "flat" cases where the same page was loaded entirely over HTTPS or HTTP.

Unsurprisingly, the downward trend of HTTP-GS content is explained by the fact that the initial HTML (and therefore all non-migrated elements) is served over HTTPS; as a result, migrating an element to HTTP-GS decreases the latency at the client. Notably, after all elements on a page are migrated from HTTPS to HTTP-GS, the end page load latency closely resembles the latency of loading the entire page over HTTP! As above, this shows that migrating content from HTTPS to HTTP-GS results in substantial performance benefits even when transparent caching is not accounted for.

7.5 Conclusion

In this chapter, we introduced a new security model for Web content delivery, *GroupSec*. GroupSec redefines the Web security model from session-based to group-based, and is the first security model to separate the privacy needs of *clients* from those of *publishers*. We provided strong arguments for why the GroupSec model

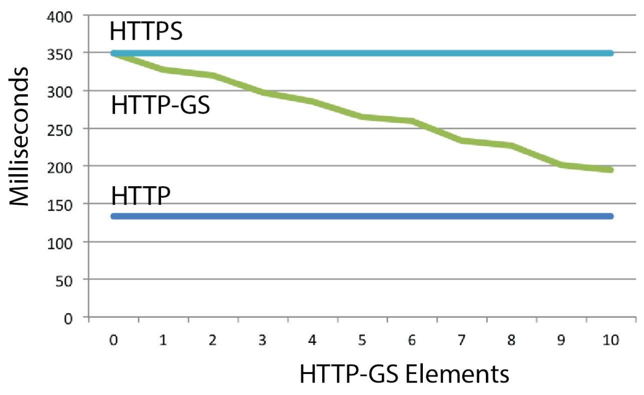


Figure 7.5: Page Load Time

better fits Web content delivery today, and enables transparent caching while still meeting the privacy needs of both clients and publishers. We also showed how HTTP can be easily adapted to GroupSec with minimal protocol changes.

GroupSec has the potential to redefine Web content delivery in a wide range of cases. More importantly, it represents a fundamental shift in how Web security is perceived. This shift invites future work and debate on a wide range of topics, including additional analysis on GroupSec-specific threats, GroupSec-related performance optimizations and integration within HTTP(S), and additional security models inspired by the GroupSec approach.

Chapter 8

Conclusion

In this thesis, we have proposed a new approach to an old problem, that of binding names, addresses, and routes. Our proposal, which is primarily based on indirection, breaks the early-bindings within the network stack and allows for dynamic and modular addressing at end hosts. We discussed how leveraging indirection between layers of the stack creates a powerful tool for solving many challenges facing networked environments today, and showed how such indirection creates a powerful tool that paves the way for future Internet architectures.

In Chapter 3, we laid the architectural framework for this proposal, and explained the design, benefits, and importance of hidden identifiers in the network stack. In Chapter 4 we implemented and evaluated HIDRA, the first network stack based on hidden identifiers, and in Chapter 5 we designed, implemented, and evaluated DIME, a new approach that leverages hidden identifier multiplexing to support IP address mobility.

Building on this foundation, we then showed how a hidden-identifier based network stack can be leveraged to support the goals of information centric networking by reusing the existing protocol stack in novel ways. In Chapter 6 we introduced a content location service that achieves the benefits of an information-centric control plane, and in Chapter 7 we introduced a new technique for secure, opportunistic blind-caching in the HTTP data plane.

In each and every chapter, we have shown that approaches based on hidden identifiers are feasible, scalable, and do not introduce significant or notable overhead at end or intermediate systems. Additionally, by concentrating our modifications at end-hosts and carefully considering the impact on intermediate systems such as routers and middleboxes, we have shown that hidden-identifier based approaches are remarkably more deployable than alternate proposals. Finally, the results of extensive experiments in many different systems show that hidden identifier approaches tend to outperform their competitors in a wide range of metrics, including latency, scalability, data-plane overhead, and control signaling, among others.

In addition to the benefits and results presented in this thesis, hidden identifier networking opens the door and paves the way for a wide range of future work. This work includes future work on the HIDRA architecture itself, such as further examination of connection-oriented protocols, network address multihoming, porting HIDRA to alternate system architectures (e.g. sensor network platforms and/or virtual machines), further optimizing network protocols to leverage hidden identifiers, extending and standardizing the HIDRA-ICN architecture, or adapting HIDRA to support service-centric networking. Additionally, there exists room for future work on integrating HIDRA-based networking as a solution to many of the challenges facing virtual networks such as V(X)LANs and other large datacenter and cloud-based network environments. Finally, the HIDRA architecture can provide a "springboard" or launching pad for other related work and projects in future network architectures, including the information- and service-centric architectures discussed throughout this thesis.

Bibliography

[ABH09] 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.

[ADI+12] B Ahlgren, C Dannewitz, C Imbrenda, D Kutscher, and B Ohlman. A survey of information-centric networking. *Communications Magazine, IEEE*, 50(7):26–36, 2012.

[ADM+08] Bengt Ahlgren, Matteo D'Ambrosio, Marco Marchisio, Ian Marsh, Christian Dannewitz, Börje Ohlman, Kostas Pentikousis, Ove Strandberg, René Rembarz, and Vinicio Vercellone. Design considerations for a network of information. In *Proceedings of the 2008 ACM CoNEXT Conference*, 2008.

[att] Global IP network latency. `http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html`.

[BB95] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for mobile hosts. *Proc. International Conference on Distributed Computing Systems*, pages 136–143, 1995.

[BBB+06] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for key management part 1: General. *NIST Special Publication*, 2006.

[BC12] M F Bari and S Rahman Chowdhury. A survey of naming and routing in information-centric networks. *Communications ...*, 2012.

[big] We Knew The Web Was Big... `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`.

[BP96] P. Bhagwat and C. Perkins. Network layer mobility: an architecture and survey. *Personal Communications*, 1996.

[BR11] E. Barker and A. Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication*, 2011.

[BS97] K. Brown and S. Singh. M-tcp: Tcp for mobile cellular networks. *ACM SIGCOMM Computer Communication Review*, pages 19–43, 1997.

[cer] So you're making an RSA key for an SSL certificate. What key size do you use? `https://certsimple.com/blog/measuring-ssl-rsa-keys`.

[CHPP12] W.K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks. *Proc. International Conference on Research in Networking*, 2012.

[CK74] V. Cerf and R. Kahn. A Protocol for Packet Network Interconnection. *IEEE Trans. Commun.*, pages 637–648, 1974.

[CK03] Edith Cohen and Haim Kaplan. Proactive caching of DNS records: addressing a performance bottleneck. *Computer Networks*, 41(6):707–726, April 2003.

[Coh08] B. Cohen. The bittorrent protocol specification, 2008.

[CS05] S. Cheshire and D. Steinberg. *Zero configuration networking: The definitive guide*. O'Reilly Media, Inc., 2005.

[DBGLA14] A. Dabirmoghaddam, M. M. Barijough, and J.J. Garcia-Luna-Aceves. Understanding optimal caching and opportunistic caching at the edge of information-centric networks. *Proc. ACM Conference on Information-Centric Networking (ICN)*, 2014.

[dig] The Slashdot Effect. `http://en.wikipedia.org/wiki/Slashdot_effect`.

[DMM08] J. Day, I. Matta, and K. Mattar. Networking is IPC: a guiding principle to a better internet. *Proc. ACM CoNEXT*, 2008.

[Dra03] R. Draves. Default Address Selection for Internet Protocol version 6 (IPv6). *IETF Standards-Track RFC 6724*, 2003.

[DVC$^+$01] 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.

[ea02] I. Stoica et al. Internet Indirection Infrastructure. *Proc. ACM SIGCOMM*, 2002.

[ea04] H. Balakrishnan et. al. A Layered Naming Architecture for The Internet. *Proc. ACM SIGCOMM*, pages 343–352, 2004.

[ea08] R. Moskowitz et. al. Host identity protocol. *IETF Standards-Track RFC 6724*, 2008.

[ea10] J. Ubillos et al. Name-based sockets architecture. *IETF Draft*, 2010.

[ea11a] A. Ghodsi et. al. Intelligent Design Enables Architectural Evolution. *ACM HotNets*, page 3, 2011.

[ea11b] T. Koponen et. al. Architecting for innovation. *ACM SIGCOMM Computer Communication Review*, 41(3):24–36, 2011.

[ea12] E. Nordstrom et. al. Serval: An end-host stack for service-centric networking. *Proc. 9th USENIX NSDI*, 2012.

[ea13] D. Farinacci et. al. The locator/ID separation protocol (LISP). *IETF Standards-Track RFC 6830*, 2013.

[Edd04] W.M. Eddy. At what layer does mobility belong? *IEEE Communications Magazine*, 42(10):155–159, 2004.

[ERB03] M. Eltoweissy, A. Rezgui, and A. Bouguettaya. Privacy on the web: Facts, challenges, and solutions. *IEEE Security and Privacy*, 2003.

[et.03] J. Crowcroft et.al. Plutarch: an argument for network pluralism. *ACM FDNA '03*, 2003.

[expa] Evil or benign? 'Trusted proxy' draft debate rages on. `http://www.theregister.co.uk/2014/02/25/evil_or_benign_trusted_proxy_draft_debate_rages_on`.

[expb] Explicit trusted proxy in HTTP/2.0 or... not so much. `https://isc.sans.edu/forums/diary/Explicit+Trusted+Proxy+in+HTTP20+ornot+so+much/17708/`.

[expc] HackerNews discussion: explicit trusted proxy in HTTP/2.0. `https://news.ycombinator.com/item?id=7296128`.

[FLT$^+$13] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce M Maggs, K C Ng, Vyas Sekar, and Scott Shenker. Less Pain, Most of the Gain: Incrementally Deployable ICN. In *Proceedings of SIGCOMM 2013*, page 1. ACM, 2013.

[FNTP12] N Fotiou, P. Nikander, D Trossen, and G C Polyzos. Developing information networking further: From PSIRP to PURSUIT. *Broadband Communications, Networks, and Systems*, pages 1–13, 2012.

[For07] B. Ford. Directions in Internet transport evolution. *IETF Journal*, 2007.

[For08] B. Ford. Breaking Up The Transport Logjam. *Proc. ACM HotNets*, 2008.

[FP] W. Ford and Y. Poeluev. An efficient certificate format for ECC. `http://csrc.nist.gov/groups/ST/ecc-workshop-2015/presentations/session2-ford-warwick.pdf`.

[FRH$^+$11] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, and Janardhan Iyengar. Architectural guidelines for multipath tcp development. *RFC6182 (March 2011), www. ietf. ort/rfc/6182*, 2011.

[FYT97] 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.

[FZ08] S. Freire and A. Zúquete. A tcp-layer name service for tcp ports. *Proc. USENIX Annual Technical Conference*, pages 275–280, 2008.

[GHJ$^+$09] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 51–62. ACM, 2009.

[GKR$^+$11] A Ghodsi, T Koponen, J Rajahalme, P Sarolahti, and S. Shenker. Naming in content-oriented architectures. *Proc. ACM SIGCOMM Workshop on Information-Centric Networking*, 2011.

[GSK$^+$11] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Information-centric networking: seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2011.

[GTW16] C. Ghali, G. Tsudik, and C.A. Wood. (the futility of) data privacy in content-centric networking. *Proc ACM CCS Workshop on Privacy in the Electronic Society*, 2016.

[GVK14] Z. Gao, A. Venkataramani, and J.F. Kurose. Towards a quantitative comparison of location-independent network architectures. In *ACM SIGCOMM Computer Communication Review*, 2014.

[Ham] J. Hamilton. The Cost of Latency. `http://perspectives.mvdirona.com/2009/10/the-cost-of-latency/`.

[Han12] D. et al. Han. XIA: Efficient Support for Evolvable Internetworking. *Proc. USENIX NSDI*, 2012.

[HF10] S. HomChaudhuri and M. Foschiano. Cisco Systems' private VLANs: scalable security in a multi-client environment, 2010.

[htta] HTTPS everywhere. `https://www.eff.org/https-everywhere`.

[httb] The HTTP Archive. `http://httparchive.org`.

[ian] Service name and transport protocol port number registry. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[IMA10] V. Ishakian, I. Matta, and J. Akinwumi. On the cost of supporting mobility and multihoming. *Proc. GLOBECOM Workshops*, 2010.

[ipv] Ipv6 celebrates its 20th birthday by reaching 10 percent deployment. `http://arstechnica.com/business/2016/01/ipv6-celebrates-its-20th-birthday-by-reaching-10-percent-deployment/`.

[JSBM02] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and the Effectiveness of Caching. *IEEE/ACM Transactions on Networking*, 10(5):589–603, 2002.

[JST$^+$09] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. Networking named content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12, 2009.

[KCC$^+$07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, 2007.

[KFV$^+$12] K Katsaros, N Fotiou, X Vasilakos, C Ververidis, C Tsilopoulos, G Xylomenos, and G Polyzos. On inter-domain name resolution for information-centric networks. *NETWORKING 2012*, pages 13–26, 2012.

[KG08] B.Y.L. Kimura and H.C. Guardia. TIPS: wrapping the sockets API for seamless IP mobility. *Proc. ACM Symposium on Applied Computing*, 2008.

[KHY12] D.M. Kline, L. He, and U. Yaylacicegi. User perceptions of security technologies. *Privacy Solutions and Security Frameworks in Information Protection*, 2012.

[KIUE00] 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.

[KP04] J. Kristiansson and P. Parnes. Application-layer mobility support for streaming real-time media. *Proc. IEEE WCNC*, 2004.

[KSB15] M. Komu, M. Sethi, and N. Beijar. A survey of identifier-locator split addressing architectures. *Computer Science Review*, 2015.

[LCP$^+$05] E.K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, 7(2):72–93, 2005.

[LFH06] 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.

[LMS$^+$14] S. Loreto, J. Mattsson, R. Skog, H. Spaak, G. Gus, D. Druta, and M. Hafeez. Explicit trusted proxy in HTTP/2.0. *IETF Standards-Track Internet-Draft*, 2014.

[LR07] R. LaRose and N. Rifon. Promoting i-safety: effects of privacy warnings and privacy seals on risk assessment and online privacy behavior. *Journal of Consumer Affairs*, 2007.

[Mah14] M. et al. Mahalingam. VXLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks. *IETF Draft*, 2014.

[MB98] D.A. Maltz and P. Bhagwat. MSOCKS: An architecture for transport layer mobility. *Proc. INFOCOM*, pages 1037–1045, 1998.

[MCD+02] Z.M. Mao, C.D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A precise and efficient evaluation of the proximity between web clients and their local DNS servers. *USENIX Annual Technical Conference*, pages 229–242, 2002.

[MWNG13] D. McGrew, D. Wing, Y. Nir, and P. Gladstone. TLS proxy server extension. *IETF Informational Internet-Draft*, 2013.

[NB09] Erik Nordmark and Marcelo Bagnulo. Shim6: Level 3 multihoming shim protocol for IPv6. *IETF Standards-Track RFC 5533*, 2009.

[NFL+14] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the S in HTTPS. *Proc. ACM CoNEXT*, 2014.

[NSV+15] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. López, K. Papagiannaki, P. Rodriguez, and P. Steenkiste. multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. *Proc. ACM SIGCOMM*, 2015.

[OMTT99] T. Okoshi, M. Mochizuki, Y. Tobe, and H. Tokuda. MobileSocket: Toward continuous operation for Java applications. *Proc. IEEE ICCCN*, 1999.

[ope] OpenSSL. `https://www.openssl.org/`.

[P. 02] P. Vixie et. al. Dynamic Updates in the Domain Name System. *IETF RFC 2136*, March 2002.

[PB15] D. Phoomikiattisak and S. Bhatti. Mobility as a first class function. *Proc. IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, 2015.

[Peo12] R. Peon. Explicit proxies for HTTP/2.0. *IETF Informational Internet-Draft*, 2012.

[Per97] C. Perkins. Mobile IP. *IEEE Communications Magazine*, 35(5):84–99, 1997.

[PGS10] L Popa, A Ghodsi, and I Stoica. HTTP as the Narrow Waist of the Future Internet. *Proc. ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[pin] Pingman: What's normal for latency? https://www.pingman.com/kb/article/what-s-normal-for-latency-and-packet-loss-42.html.

[PJ96] C. Perkins and D.B. Johnson. Mobility support in IPv6. *Proc. 2nd International Conference on Mobile Computing and Networking*, pages 27–37, 1996.

[PPKC06] S. Pack, K. Park, T. Kwon, and Y. Choi. SAMP: scalable application-layer mobility protocol. *IEEE Communications Magazine*, 44(6):86–92, 2006.

[PSS04] E. Perera, V. Sivaraman, and A. Seneviratne. Survey on network mobility support. *Proc. ACM SIGMOBILE*, 2004.

[PV11] Diego Perino and Matteo Varvello. A reality check for content centric networking. *Proc. ACM SIGCOMM Workshop on Information-Centric Networking*, 2011.

[RCC$^+$11] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. Tcp fast open. *Proc. Conference on Emerging Networking Experiments and Technologies (CONEXT)*, 2011.

[Rip01] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. *Proc. International Conference on Peer-to-Peer Computing*, pages 99–100, 2001.

[RL16] J. Reschke and S. Loreto. 'Out-Of-Band' content coding for HTTP. *IETF Standards Track Internet-Draft*, 2016.

[RLC05] N. Rifon, R. LaRose, and S. Choi. Your privacy is sealed: Effects of web privacy seals on trust and personal disclosures. *Journal of Consumer Affairs*, 2005.

[RPB$^+$12] 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.

[RS04] V. Ramasubramanian and E.G. Sirer. The design and implementation of a next generation name service for the Internet. *ACM SIGCOMM Computer Communication Review*, 34(4):331–342, 2004.

[rsa] RSA laboratories: What key size should be used? `http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/key-size.htm`.

[RT78] D. Ritchie and K. Thompson. The unix time-sharing system. *The Bell Systems Technical Journal*, 1978.

[Sal93] J. Saltzer. On The Naming and Binding of Network Destinations. *RFC 1498*, August 1993.

[SB00] 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.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. *RFC 3550*, July 2003.

[SGLA15] 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.

[Sho78] J. Shoch. Inter-Network Naming, Addressing, and Routing. *17th IEEE Computer Society Conference (COMPCON 78)*, 1978.

[SLPR15] J. Sherry, C. Lan, R. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *Proc. ACM SIGCOMM*, 2015.

[SMGLA13] S. Sevilla, P. Mahadevan, and J.J. Garcia-Luna-Aceves. FERN: A unifying framework for name resolution across heterogeneous architectures. *Proc. IFIP NETWORKING*, 2013.

[SMMC04] 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.

[Sri11] M. et al. Sridharan. NVGRE: Network virtualization using generic routing encapsulation. *IETF Draft*, 2011.

[SSII02] 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.

[sta] How big an RSA key is considered secure today? `http://crypto.stackexchange.com/questions/1978/how-big-an-rsa-key-is-considered-secure-today`.

[STA01] Anees Shaikh, Renu Tewari, and Mukesh Agrawal. On the effectiveness of dns-based server selection. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1801–1810. IEEE, 2001.

[STU$^+$14] 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.

[SW00] H. Schulzrinne and E. Wedlund. Application-layer mobility using SIP. *Mobile Computing and Communications Review*, 4(3):47–57, 2000.

[TBD$^+$11] J. Touch, I. Baldine, R. Dutta, G.G. Finn, and B. Ford. A dynamic recursive unified internet design (DRUID). *Computer Networks*, 2011.

[TEH16a] M. Thomson, G. Eriksson, and C. Holmberg. An architecture for secure content delegation using HTTP. *IETF Standards Track Internet-Draft*, 2016.

[TEH16b] M. Thomson, G. Eriksson, and C. Holmberg. Caching secure HTTP content using blind caches. *IETF Standards Track Internet-Draft*, 2016.

[TGDM11] E. Trouva, E. Grasa, J. Day, and I. Matta. Transport over heterogeneous networks using the RINA architecture. *Wired/Wireless Internet Communications*, 6649(Chapter 25):297–308, 2011.

[Tho16] M. Thomson. Encrypted content-encoding for HTTP. *IETF Standards Track Internet-Draft*, 2016.

[TP08] J.D. Touch and V.K. Pingali. The RNA metaprotocol. *Proc. International Conference on Computer Communications and Networks*, pages 1–6, 2008.

[TZY01] C.W. Turner, M. Zavod, and W. Yurcik. Factors that affect the perception of security and privacy of e-commerce web sites. *International Conference on Electronic Commerce Research*, 2001.

[VBZ$^+$12] T. Vu, A. Baid, Y. Zhang, T. Nguyen, J. Fukuyama, R. Martin, and D. Raychaudhuri. Dmap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet. *Proc. IEEE ICDCS*, 2012.

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

[Wat81] R.W. Watson. Identifiers (Naming) in Distributed Systems. *Distributed Systems–Architecture and Implementation (LCN 105)*, Chapter 9:191–210, 1981.

[WB13] S. Weiler and D. Blacka. RFC 6840: Clarifications and Implementation Notes for DNS Security (DNSSEC). *IETF Standard*, 2013.

[WS99] E. Wedlund and H. Schulzrinne. Mobility support using SIP. *Proc. ACM WoWMoM*, 1999.

[XVT$^+$12] G Xylomenos, X Vasilakos, C Tsilopoulos, V A Siris, and G C Polyzos. Caching and mobility support in a publish-subscribe internet architecture. *Communications Magazine, IEEE*, 50(7):52–58, 2012.

[XY13] X. Xu and L. Yong. NVGRE and VXLAN encapsulation extension for L3 overlay. *IETF Draft*, 2013.

[ZM02] V. Zandy and B. Miller. Reliable network connections. *Proc. ACM MOBICOM*, 2002.