AINA 2014 - IEEE 28th Intl. Conf. Advanced Information Networking and Applications. Victoria, Canada. May 2014.

An identifier-locator approach to host multihoming

Bruce Simpson
University of St Andrews, UK
bs48@st-andrews.ac.uk

Saleem N. Bhatti
University of St Andrews, UK
saleem@cs.st-andrews.ac.uk

***Abstract*—Host multihoming allows individual hosts to be multiply connected to the network, e.g. by concurrent use of two network prefixes, each network prefix tied to a separate network interface. Such multihoming capability improves the host's ability to implement such features as load-balancing, fail-over and multi-path transport protocols. However, IP does not directly support host multihoming today. The *Identifier / Locator split* solution space is seen as one way for reducing such negative impact. We present an evaluation of host-multihoming as a prototype implementation of the *Identifier Locator Network Protocol (ILNP)* on FreeBSD, as a superset of IPv6 – called ILNPv6. We demonstrate load-balancing using ILNPv6 multihoming and compare performance with IPv6 forwarding at the end host.**

I. Introduction

The growth of the Internet routing table is a significant problem. It is known that *load balancing* and *multihoming* are the two principal factors involved. Whilst much of this growth is at the edge [1]–[4] of the network, it impacts upon the core network: the state required for these functions must be propagated throughout the routing infrastructure.

There is significant overlap between *node identity* and *network location* as represented by the way IP addresses have been used for 35 years [5], [6]. Architectures which propose a separation of location from identity may mitigate the effects of routing table growth [1]. Whilst IPv6 has expanded the address space, the routing and addressing architecture has not changed to reflect location independence. The *Identifier-Locator Network Protocol (ILNP)* is a host-based, end-to-end 'ID/Loc' architecture. ILNP nodes do not use IP addresses but instead use *Node Identifier* and topologically-significant *Locators*.

A multihomed host (or site) is connected to more than one IP network. With IP today, multihoming and load balancing must be implemented within the routing infrastructure. In 'ID/Loc' architectures, these may be realised as a choice between network locations. ILNP advertises network location values in the Domain Name System (DNS) [7]. Load balancing can be viewed as a choice between these locations (based on an additional policy), and therefore may be implemented without requiring support from network routers or 'middleboxes'. It is, effectively, a policy applied to the forwarding plane.

A. Contribution

While ILNP is a radically different architecture, the claim is that it can be implemented on the current codebase, rather than requiring a 'clean-slate' approach. We examine how the FreeBSD stack can be modified to implement ILNPv6 and demonstrate multihoming and load balancing functions. We evaluate its performance in comparison to IPv6. We use the terms *Locator* and *Identifier* as defined in RFC6115 [8].

We begin in Section II by describing how an 'ID/Loc' split affects routing state for multihomed sites (and hosts). After presenting some related work in Section III, we describe how the ILNPv6 host architecture has been implemented in Section IV, with emphasis on how existing Application Programming Interfaces (APIs) are affected. Section V describes our experimental configuration, method and results. Scalability and performance is discussed briefly in Section VI, and we conclude in Section VII.

II. Background

The overloaded use of IP addresses today may be understood by considering the bindings of an IP address within the protocol stack [9], as shown in Table I. The second column shows that IP addresses are used as both node *identifiers* and topological *locators*. Whilst this table describes these in terms of 'names' we do not discuss this beyond API compatibility.

TABLE I
Use of names in IP and ILNP

Protocol layer	IP (v4 and v6)	ILNP (ILNPv6)
Application	FQDN*, IP address	FQDN or app-specific
Transport	IP address	(node) Identifier (NID), *NID*
Network	IP address	Locator (L64), *L64*
(interface)	IP address	dynamic binding

* Fully Qualified Domain Name

The IP network layer and routing functions use the IP address to identify IP sub-networks, i.e. network *location*. Such use of IP addresses as *locators* requires that routers exchange information about IP address prefixes and the paths by which they may be reached, e.g. using routing protocols such as the Border Gateway Protocol (BGP).

Transport layer communication protocols (such as TCP and UDP) also use the IP address to *identify* nodes. In conjunction with port numbers, these are used to uniquely identify sessions between two hosts. However, as can be seen in Table I, the overloaded use of the IP address creates an implicit binding. Transport layer sessions become 'bound' to specific interfaces as they are established, making it difficult to separate location from identity within the current IP architecture. Host-based approaches to the 'ID/Loc' split aim to disentangle these two uses, which we discuss further in Subsec. IV-A and IV-J.

Multi-homing may be desirable for hosts, e.g. to achieve robustness through diverse connectivity. However, it needs special treatment for IP [10]. Moreover, it contributes to routing table growth by requiring routers to advertise additional prefixes. This can be demonstrated by considering Fig. 1. Here, a site network uses two prefixes, P_1 and P_2. These must both be visible upstream from the two ISPs, ISP1 and ISP2.

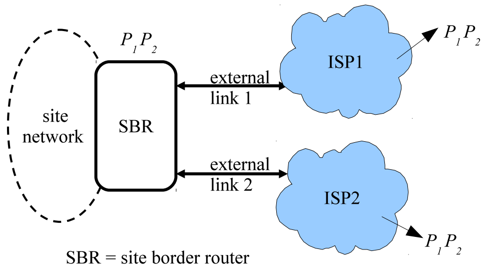


Fig. 1. Illustrative scenario for multihoming: a site network, with a site border router (SBR), connecting to two separate ISPs. We assume that, in this case, the site network has two routing prefixes, P_1 and P_2.

We assume that these prefixes are *de-aggregated* within the global routing table, i.e. that connectivity to each ISP is topologically diverse (common practice where BGP is used to multihome sites). Such prefixes are also known as *Provider Independent (PI)* prefixes, allocated to allow the user (the end site) to maintain use of these prefixes even if their ISP changes. Therefore, ISP1 and ISP2, as well as upstream routers must advertise these prefixes separately. This is because they form part of the IP address, which is used in end-to-end state (identity) for hosts – see Table I. It follows that the additional upstream routing state required for N_P site prefixes with N_I upstream ISPs is $O(N_P.N_I)$.

Whilst we have used site multihoming in our example (as that is the more common case for IP today), a site would need to support multihoming to allow individual hosts to be multihomed, and the scalability analysis is the same.

III. Related work

A excellent summary of approaches to the 'ID/Loc' split is provided in RFC6115 [8]. These may be divided into two categories: *network-based* approaches which make the split a property of the routing infrastructure (leaving IP addresses unchanged at hosts), and *host-based* approaches which make it a property of host address management (possibly introducing new namespaces or addresses). In line with the IETF's current thinking regarding the 'sunsetting' of IPv4 [11], many of these consider only IPv6. So, we constrain ourselves here to consider three proposals with similar intent and scope to those of ILNP.

The *Host Identity Protocol (HIP)* [12] supports multihoming and mobility. The 'ID/Loc' split is implemented using an asymmetric key crypto-system; Host Identifiers (HI) are public keys, Locators are IP addresses. Whilst a host can have many HIs, each HI must identify a single host. HIs may exist in public and private namespaces; they are never directly exposed to the routing system, nor are they directly represented by addresses. DNS maps a host name to a HI entry, using the HIP Resource Record [13]. Modifications are required for applications and transport protocols, using separate APIs to expose HIP functionality. Applications must use HIs in place of network addresses. Compared to ILNPv6, HIP poses an additional requirement for use of cryptographic identities, which may not be suited for all uses. Additionally, HIP requires existing applications to be ported to a new API, while ILNP can work with the existing sockets API (see later).

Level 3 Multihoming Shim Protocol for IPv6 (SHIM6) [14] supports multihoming and traffic engineering for IPv6. SHIM6 is host-based and end-systems use two IPv6 address – one as an Identifier and one as a Locator. It is implemented as a shim underneath IPv6 transport protocol APIs. A four-way handshake is used to establish a SHIM6 context (exchange of Identifier and Locator values), and separate security mechanisms are required to protect its signalling [15], [16].

The *Locator/Identifier Separation Protocol (LISP)* [17] proposes a complete routing architecture and is not host based. This has the advantage that end-host stacks do not need to change. LISP's Routing Locators (RLOCs) and Endpoint Identifiers (EIDs) re-use IP address values. LISP requires an in-network mapping system between the two namespaces. LISP uses a 'map-and-encapsulate' (map-encap) approach to forwarding (i.e. tunnelling), creating an overlay mesh. Additional headers are interposed between existing IPv4 header fields, which are normally visible only to LISP-enabled routers. LISP may require significant changes to some existing network routers (perhaps the deployment of new routers). The map-encap mechanism may impact existing applications, e.g. due to MTU size issues, and use of tunnels.

HIP, SHIM6 and ILNP are host-based 'ID/Loc' architectures. These approaches push deployment costs mainly to the end-user, though there may be impact to site networks, e.g. firewall configuration for new extension headers. Whilst ILNP requires changes to the end-host stack, no shim layer is required. It does not require additional IPv6 addresses for signalling its protocol state. ILNP does not use tunnelling.

IV. ILNPv6 Overview

ILNP defines an 'ID/Loc' split architecture with experimental RFC status [9], [18]. It is specified for both IPv6 [19], [20] and IPv4 [21]–[23] networks, i.e. ILNPv6 can be engineered as a set of extensions to IPv6. In this paper, we discuss only those aspects which apply to IPv6 hosts. As, architecturally, ILNP is radically different from IP – there are no addresses in ILNP – our implementation considers (1) what needs to change in current stack engineering in order to enable ILNPv6; and (2) what impact this could have on performance. This work has taken a dual-stack approach to implementation, as described further in RFC6740 [9] and RFC6741 [18].

Our implementation is based on FreeBSD 8.2, and is a separate kernel module, `ilnp6`, with changes to the existing IPv6 stack. IPv6 applications may use ILNPv6 without changes (described in Subsec. IV-I). Multihoming, fail-over, and load- balancing are also transparent to IPv6 applications (described in Subsec. IV-H).

A. Identifier-Locator Vectors

The term *Identifier-Locator Vector (I-LV)* [9, Sec. 3.3] refers to an ID/Loc pairing encoded into an IPv6 packet, as shown in Fig. 2. This is key to understanding how existing IPv6 functionality may be re-used to implement ILNPv6, including the address assignment mechanisms. The *L64* value is an existing IPv6 routing prefix and has the same semantics. *So, no changes to core network routers are required to forward ILNPv6 packets.* The *NID* value is generated as for IPv6, but interpreted differently in the end-system stack.

```
IPv6 Address (RFC3571 + RFC4291):
  |             64 bits              |          64 bits              |
  +----------------------------------+-------------------------------+
  |      Unicast Routing Prefix      |    Interface Identifier       |
  +----------------------------------+-------------------------------+

ILNPv6 Identifier-Locator Vector (I-LV) (RFC6741):
  |             64 bits              |          64 bits              |
  +----------------------------------+-------------------------------+
  |           Locator (L64)          |    Node Identifier (NID)      |
  +----------------------------------+-------------------------------+
```

Fig. 2. IPv6 unicast format with ILNPv6 unicast format. The ILNPv6 Locator has the same syntax and semantics as that of an IPv6 routing prefix. The ILNPv6 Identifier has the same *syntax* as an IPv6 Interface Identifier, but its *semantics* are that of *node* identity and not an *interface* identity.

One motivation for the re-use of IPv6 address bits is that no changes are necessary to any IPv6 network elements, other than those end-hosts which participate directly in the ILNPv6 protocol. For IPv6 hosts, the scope of our changes is limited to a subset of the networking stack. Therefore, IPv6 and ILNPv6 packets ‘on-the-wire’ are largely identical; we discuss their differences below in Subsec. IV-F.

B. Local identifiers

Our implementation re-uses the existing IPv6 interface identifier (*IID*) mechanism of RFC4291 [24, Sec. 2.5.1]. This allocates a unique *IID* for each interface based on hardware address. ILNPv6 re-uses the *IID* values as node identifier (*NID*) values [9, Sec. 3.1]. A *NID* value need only be unique within the scope of its Locator, but using existing IPv6 mechanisms, we achieve the same level of uniqueness for *NID* values as for IPv6 *IID* values. Where such an identifier is not available, the behaviour is implementation specific, although the IPv6 address (or I-LV for ILNPv6) must be unique within a subnet prefix [24, Appendix A].

For convenience (to aid development and debugging), we modified the FreeBSD kernel to use some bits from an MD5 digest of the host name.[1] The same *NID* value can be used across all interfaces in ILNPv6.

C. Routing prefixes for locator values

In order to discover local *L64* values, ILNPv6 re-uses the existing IPv6 Router Advertisement (RA) mechanism. The format of RA messages and their handling by the IPv6 stack are unchanged, however we have added notifications of prefix-related events (e.g. advertisements, expiry of existing prefixes, or explicit withdrawal of a prefix by an on-link IPv6 router). When the prefix list changes, `ilnp6` updates the local list of *L64* values.

[1]IPv6 stacks derived from KAME ‘fall back’ to this behaviour, where no hardware-based identifier is available.

D. Locator precedence

Locators have an additional property: *precedence*. This is represented by a 16-bit unsigned integer value, with lower values being preferred. Precedence allows multiple *L64* values to be used simultaneously, and interpreted as local policy dictates. This property is not native to IPv6, therefore it must be added to the network stack. During system initialisation, default precedence values are loaded from a local policy table. The table is managed using a new system command, `ilnp6locctl`. The implementation was derived from FreeBSD’s `ip6addrctl` tool, which manages IPv6 default address selection policies [25].

E. Discovering peer I-LVs

ILNP applications are intended to use names, whereas IP applications must use addresses. However, a node initiating an ILNPv6 session must first learn a remote ILNPv6 node’s *L64* and *NID* values [18, Sec. 6]. In the ILNP architecture, both may be advertised in the existing Domain Name System (DNS). Moreover, existing naming APIs may be modified to support ILNP. New DNS RRs have been defined for ILNP [7], and commercial DNS implementations now support these RRs.[2] ILNP’s *NID* and *L64* DNS RRs are analogous to the `AAAA` resource record (RR) used for IPv6 addresses.

A host’s ‘stub resolver’ may be configured to use the `/etc/hosts` file to resolve names. This may be for bootstrapping or as a fallback measure when DNS is unavailable, or through explicit configuration by the host’s administrator. For IP, this file contains a static mapping of names to network addresses, and is required during system initialisation. We have extended the `/etc/hosts` syntax to include a 1:M mapping of names to I-LVs, and this is summarised in Fig. 3. (Our evaluation in Section V uses `/etc/hosts`.)

```
#
# /etc/hosts file extended syntax for ILNPv6
#
# L64       64-bit Locator value (in IPv6 address format)
# lprec     the Locator's precedence value
# NID       64-bit Node Identifier value (in Canonical EUI64 format)
# hostname  a valid hostname value
#
# An entry -- an I-LV record -- has the structure:
#
#   L64|lprec,NID     hostname
#
# Example entries are:

2001:0db8:d00d:0000|10,02-1f-5b-ff-fe-ff-13-74   foo.yoyodyne.com
2001:0db8:cafe:0000|20,2a-37-37-ff-fe-1c-cf-fe   bar.yoyodyne.com
```

Fig. 3. The extended syntax in `/etc/hosts` for ILNPv6.

IPv6 applications typically use the portable `getaddrinfo()` API to resolve names to addresses; it is specified in RFC3493 [26] as being protocol independent.[3] Backwards compatibility is discussed below in Subsec. IV-I. The `getaddrinfo()` API returns a linked-list of `addrinfo{}` records, and *may* sort these records using the prefix selection criteria in RFC6724 [25]. However, IPv6

[2]At the time of writing NLNetLabs *NSD v3.2.15* and ISC BIND 9.9.3/9.8.5/9.6-ESV-R9 support RFC6742.

[3]Other naming APIs offer only a subset of `getaddrinfo()` functionality (or are platform specific), therefore we do not discuss them here.

applications might discard records beyond the first result, depending on how they are written. A request for an ILNP name may return multiple *L64* values. We have modified the API to return all matching values from /etc/hosts in order of locator precedence (not using RFC6724 criteria).

F. Session initiation and security

Having discovered a peer's identifier (*NID* value) and locator set (*L64* values), the node may then initiate an ILNPv6 session with that peer. ILNPv6 defines a new IPv6 destination option: the *Nonce Option* [20], Fig. 4. It is defined for use only by ILNPv6.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   | Hdr Ext Len   |Opt. Type 0x8b | Option Length |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                          Nonce Value                          /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 4. IPv6 Destination Option containing ILNPv6 Nonce (RFC6744)

The Nonce Option contains a 32-bit (or 96-bit) value unique to each session (and peer), derived using the MD5 hash algorithm. The first packet of an ILNPv6 session must include this option in its headers. However, it is not required in all session packets. This option has two roles. It flags a packet as being an ILNPv6 packet and enables a handshake for ILNP session initiation. It also protects against off-path attacks on ILNPv6 sessions. If such basic security is insufficient, then the use of IPsec is recommended [27, Sec. 4.4], with Identifier values used as part of the IPsec Security Association in place of IP addresses.

G. Locator updates

Once an ILNPv6 session has been established, no further DNS lookups are necessary. Sessions use the ICMPv6 Locator Update (LU) [21] message to communicate further changes between peers (e.g. new connectivity, explicit withdrawals, and changes in locator precedence). LU messages *must* include the Nonce Option described above. Events which modify the Locator set (described in Subsec. IV-C) will cause LU messages to be sent by the ilnp6 module. Should intermittent changes occur, these events are filtered within a time window of 500ms to prevent bursts of redundant LU messages.

H. Locator selection

ILNP nodes may have many *L64* values. The protocol does not impose any defined behaviour on *locator selection* (i.e. how a specific I-LV is resolved to the final IPv6 destination during transmission). This flexibility permits several network layer functions to be implemented at the host, without requiring 'middleboxes' (e.g. multihoming and load-balancing). Site administrators may define *locator selection policies* to achieve the desired behaviour, using the precedence table described in Subsec. IV-D.

To demonstrate this capability (and how locator selection is intended to function more generally), we implemented a simple per-packet volume (load) balancing algorithm (Fig. 5). Each ILNPv6 packet transmission invokes this procedure. The *L64* values for the packet's source and destination (s_ilv, d_ilv) are rewritten [28, Sec. II-G] according to the state of the locator(s) known to the session S. Whilst IPv6 datagrams are not affected, the existing IPv6 output path is later re-used by ILNPv6 transmission.

To re-use existing code, remote and local I-LVs are 'mapped' to IPv6 addresses, in ascending order of precedence. Remote locators reside in a per-session tail queue, as a Correspondent Cache [18, Sec. 5.4]. This choice was made for ease of implementation, rather than performance.

Fig. 6 shows a combined call graph for transmit/receive of ICMPv6 ECHO messages. The locator selection function ilnp6_sess_select_ilvs() is highlighted in reverse type, data plane in white, and control plane in yellow. Dashed lines indicate IPv6 specific code paths. The FreeBSD kernel function in6_selectsrc() looks up an output interface and a source address from a provided IPv6 destination (the host's Forwarding Information Base (FIB) is used during this process). Locator selection uses this function to resolve each candidate *I-LV* (i.e. up to $N+1$ times, where N is the number of active remote *L64* values within the session). Unreachable locators are rejected and are not used for volume balancing. Finally, the selected *L64* value is resolved to the *source I-LV*.

I. Compatibility with IPv6 applications

It would be desirable to allow IPv6 applications to use ILNPv6 without changes.[4] To do so, we use a *lookaside cache* in our implementation. The cache 'maps' between the application's use of an *I-LV* (as if it were an IP address provided to the sockets API), and the *I-LV* set resolved from a name corresponding to an ILNP node (discovered within the naming API).

The naming APIs and sockets APIs are orthogonal to each other in terms of how they are used (and often their implementation, also). Consider that a typical IPv6 application (e.g. ssh or a World Wide Web browser) will resolve a name to an IPv6 address using getaddrinfo(), and establish a connection to a remote site by using the connect() function from the sockets API. This function accepts network addresses, not names, i.e. sockets are not visible to getaddrinfo(), and names are not visible to connect().

In FreeBSD the socket API is part of the kernel, whereas the naming API is part of the C runtime library, and cannot directly manipulate sockets. Therefore, a 'downcall' to the kernel is required. We use the sysctl API [29] to populate the lookaside cache when an *I-LV* is resolved from a name. The calling thread and process ID is recorded with each *I-LV* to prevent inadvertent use of ILNPv6 by other applications.

When a socket API is invoked, the lookaside cache is examined for an *I-LV* value corresponding to the IPv6 address passed by the application. If there is a cache 'hit', the first socket API call which matches the *I-LV* will use an ILNPv6 session (allocated on demand), and the socket will transition

[4]Backwards compatibility between IPv6 and ILNPv6 hosts is described in RFC6740 [9, Sec. 8].

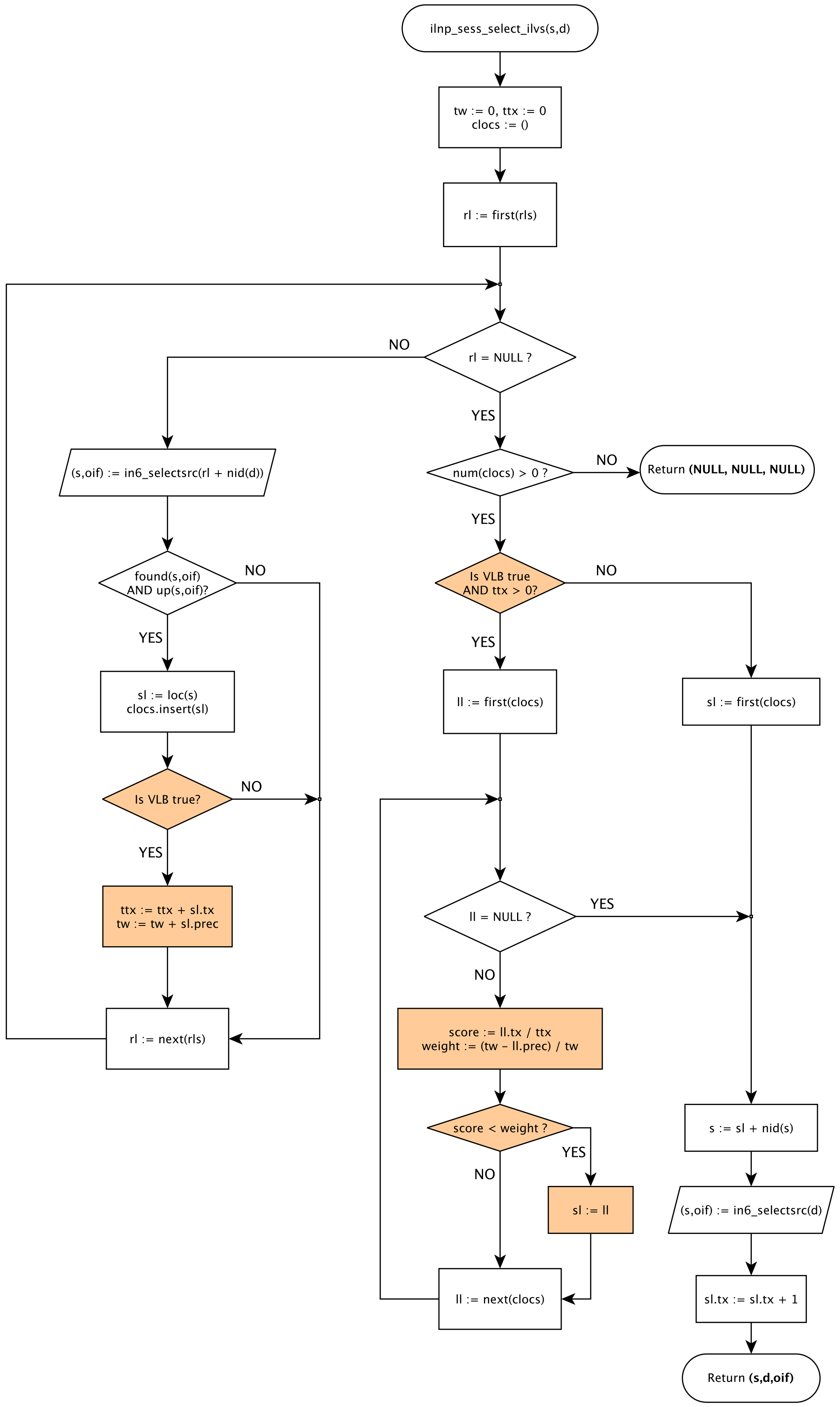


Fig. 5. Flowchart for the *locator selection procedure* (the reverse-type box marked `ilnp6_sess_select_ilvs` in Fig. 6). The plain (white) boxes steps are common to all ILNP sessions. The shaded (orange) boxes are specific to volume (load) balancing, where the boolean variable `VLB` will be set to TRUE. Parameters `s` and `d` denote the packet's source and destination I-LVs. Each session has a set of locators `rls` advertised by the remote peer. The set of candidate local locators `clocs` is ordered by their precedence `prec`, each with a count of transmitted packets `tx`. The variable `oif` points to the output interface chosen by the host's IPv6 source selection function `in6_selectsrc()`. Calls to this function are shown in slanted rectangles. `loc()` and `nid()` refer to the upper and lower 64 bits of an I-LV. Concatenation of L64 and NID values to form an I-LV is also denoted by the + symbol.

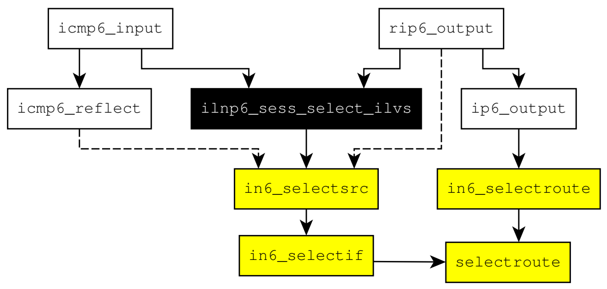


Fig. 6. Call graph for ICMPv6 transmit and receive in ILNPv6. The reverse-type (black) box contains the *locator selection procedure* (shown in Fig. 5). It performs control-plane and data-plane functions. The shaded (yellow) boxes are unmodified IPv6 control-plane functions. The plain (white) boxes are IPv6 data-plane functions. Dashed lines indicate IPv6-only code paths.

to ILNPv6 operation. Entries persist for 1000ms after the last socket API call, and IPv6 binaries are unchanged.

Alternative approaches (e.g. system call wrappers or `setsockopt()` calls) would require changes to the API (and, by extension, applications). However, such API changes may be beneficial for new ILNPv6 applications to exploit enhanced functionality and control in the future.

J. Location-independent sockets

ILNP sockets are independent of network location, and transport protocols use only *identifiers* in the binding of their endpoints. However, in the present IP architecture, sockets may become bound to specific network locations as a side effect of how IP addresses are used (Table I shows that it is used as an identifier). This use is pervasive and historical [30].

Transport protocols (e.g. TCP, UDP) differentiate between sockets by combining the network address and port number for both local and remote endpoints, forming the *4-tuple*. When the `bind()` or `connect()` APIs are invoked, the host will look up the interface used to reach the remote peer in the host's FIB, and select an appropriate source address.

Whilst the binding is to the IP address only, this implies that sockets are bound to a specific interface (i.e. the interface used to reach the next hop to the destination), as the source address cannot be changed without disrupting existing connection state. For ILNP sockets, the structure of the 4-tuple does not change, although each address in the 4-tuple is interpreted as an *I-LV* , and *L64* values may be rewritten by locator selection when a packet is transmitted.

Our evaluation in the next section uses ICMPv6, and therefore raw IPv6 sockets, which have simpler binding semantics (they are 'bound' only in terms of an IP protocol number). The changes required for TCP and UDP are for future work.

V. Performance Evaluation

We perform a *comparative* evaluation against IPv6 and *not an absolute performance evaluation*. This is in keeping with our aim to see how ILNP can be engineered into an existing codebase. We evaluated our implementation in the VMware Fusion v5.0.1 virtual machine monitor, running on a MacPro (2x Quad-Core Xeon, 16GB RAM). Each VM image was allocated 1 core and 1GB of RAM. This was our development environment, which was convenient, but, in related work, we are also interested in looking at ILNPv6 use in data-centre scenarios [31], [32]. The behaviour of VMware Fusion itself has not been profiled during this work. Rudimentary observations of its possible impact on performance were made using `iperf`, but are not reported here. Our analysis does not include DNS lookups, as both ILNPv6 and IPv6 applications perform these before session initiation.

A. Configuration

We tested the multihoming implementation with two interfaces, each with a separate Locator value. Fig. 7 illustrates the experimental network topology. Each node had multihomed connectivity to two 'vmnet' interfaces, to simulate Ethernet sub-networks with physical separation.

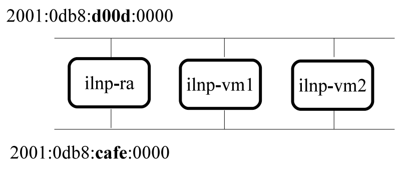


Fig. 7. ILNPv6 experimental topology. All nodes are VM images running in VMware Fusion. Node `ilnp-ra` ran an unmodified installation of FreeBSD 8.2, and was configured as an IPv6 router without forwarding. Its only role was to advertise the two network prefixes shown ('cafe' and 'd00d'), using standard IPv6 Router Advertisements (RAs). Nodes `ilnp-vm1` and `ilnp-vm2` ran a modified FreeBSD with ILNPv6 enabled. Network prefix configuration was obtained dynamically from the IPv6 RA.

B. Method

We performed a quantitative performance study of the ILNPv6 prototype, using `bwping6`.[5] It transmits and receives ICMPv6[6] ECHO streams, using a simple timed delay loop with three parameters: goodput threshold, ICMPv6 ECHO payload size, and total ICMPv6 traffic volume. Changes were made to report *mean RTT* and standard deviation, in a format suitable for processing by the *R* statistical software package. We also verified that locator selection (and volume balancing) operated correctly by modifying the `fping6`[7] utility to output per-interface (per Locator) statistics. No ILNP-specific code was added to `bwping6` or any external tools.

C. Results

We ran IPv6 streams between unmodified FreeBSD kernels, the 'IPv6' result group, as a benchmark against the ILNPv6 results groups. Measurements were performed for three experimental factors: Ethernet frame size (128 bytes and 1514 bytes); target goodput (1, 10, 20, 30, 40, 50 and 100 Mbps); and load (volume) balancing mode (Locator preference values of: 100/0 i.e. one interface is not used; 50/50 i.e. equal split; and 80/20 i.e. unequal split). 5 trials were performed for each factor group with results at 95% confidence (error bars may

[5]http://bwping.sourceforge.net/

[6]Two limitations prevent the use of `ping6`, `iperf`, and `netperf`: no goodput threshold parameter, and no support for ICMPv6 packet generation.

[7]http://fping.org/

be too small to be visible in some cases), and a minimum trial duration of 120 seconds. Mean RTT figures for all factors are illustrated in Fig. 8a and Fig. 8b, with a 1-2-5 log scale. The dashed lines are a visual aid to show trends in the data. Finite sample correction was applied at higher data rates.

Volume balancing was evaluated by measuring the distribution of received ICMPv6 ECHO responses at two interfaces (locators) where ILNPv6 was in use. During each trial, 1000 ICMPv6 ECHO requests were sent by `ilnp-vm1` to `ilnp-vm2` over a 100s interval (i.e. a delay of 100ms between requests). 5 trials were performed for each group, with results at 95% confidence (loss is not visible at this scale).

The choice of the minimum frame size accounts for the presence of the 32-bit Nonce Option (8 bytes). Whilst not required in all packets, our evaluation does so for convenience. The payload size was held constant within factor groups by adding 8 bytes to the payload in IPv6 groups. For 128 byte frames, the graph in Fig. 10 is derived from the group means of measured goodput across trials, weighted by elapsed wallclock time, and divided by requested goodput.

TABLE II
EXPERIMENT GROUPS WITH MEAN PACKET LOSS >100PPM

Frame (Bytes)	Mbit/s	Configuration	% Loss
128	100	ILNPv6, 50/50	48.37
128	100	ILNPv6, 80/20	48.37
128	100	ILNPv6, 100/0	48.30
128	100	IPv6	41.79
128	20	ILNPv6, 100/0	0.15
128	20	ILNPv6, 50/50	0.12
128	20	ILNPv6, 80/20	0.02

VI. DISCUSSION

Fig. 9 shows that volume balancing – and therefore, locator selection – functions as expected. This demonstrates one application of ILNPv6 (and not IPv6). Responses are distributed within 0.1% of the requested weighting by precedence value. However, we would expect a drop in performance for ILNPv6 compared to IPv6, as additional processing is required to enable the use of multiple Locator values and load-balancing. (These are preliminary results from an unoptimized implementation of ILNPv6.)

Both Fig. 8a and Fig. 8b show a fairly constant increase in average RTT across ILNPv6 trials. Table II shows that loss exceeded 48% for 128 byte frames at 100Mbit/s, for all ILNPv6 factor groups (41% for IPv6). These were excluded from further analysis. Loss for other factor groups was not considered significant. We observed the same effect when running `iperf` UDP sessions with identically sized frames. Therefore, we believe that the VMware Fusion software switch is being saturated at rates exceeding 10K packets-per-second (pps). This has limited goodput to 18.8Mbit/s in the affected trials, but impacts IPv6 and ILNPv6 equally.

Measured goodput for 1514 byte frames was not significantly affected by ILNPv6. However, an overall reduction in goodput was observed across all groups for 128 byte frames beyond 10Mbit/s, as shown in Fig. 10. The volume of traffic for each rate is fixed, and the load balancing algorithm is a simple volume balancer. Therefore, the similar goodput we observed across factor groups is expected.

Qualitatively, we believe that the higher average RTT results and impeded goodput for 128 byte Ethernet frames are due to the additional per-packet processing overhead of locator selection (described in Subsec. IV-H). We reiterate that this must be performed for each individual packet transmission, and it is likely that the redundant FIB lookups have degraded performance. (A detailed systems-level profiling of this behaviour is for future work.)

A. Scalability

ILNPv6 may also be used for site multihoming [33]. The mechanisms for site multihoming are almost identical to those used for host multihoming: a site border router provides a convenient management point for optimising the management of a site-wide function [31]. We have focused on modelling host behaviour as this allows experiments to be constructed more readily. However, the scalability analysis of routing state is the same as for sites.

As discussed in Section II, no additional routing state is required for ILNPv6. However, the Locator values are stored in DNS, so there is a state displacement from routing state to DNS state. The DNS state is $O(1)$ in a single point in the DNS, compared to $O(N_P.N_I)$ state for all upstream routers as is the current situation for IP. Moreover, the DNS lookups will only be performed during session setup. This behaviour is independent of the network layer (as is the case with IPv6), therefore we have not studied it here.

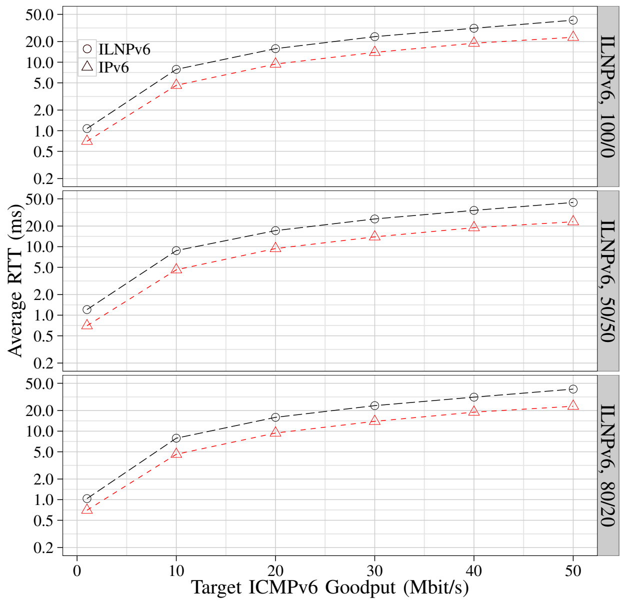
B. Performance

We have identified ways in which performance can be improved.[8] The re-use of IPv6 functions has minimized engineering change, but assumptions about the semantics of IPv6 addresses pervade the networking stack, creating engineering challenges. For example, the `in6_selectsrc()` function is normally used to resolve the source address of a socket during `connect()` or `bind()` operations. These operations normally take place only once during its lifetime. Each call to `in6_selectsrc()` may acquire up to 6 locks (mutexes) to serialize access to its required data structures. So, whilst `in6_selectsrc()` is not the optimal solution, it shows that existing IPv6 code can adapted for ILNPv6.

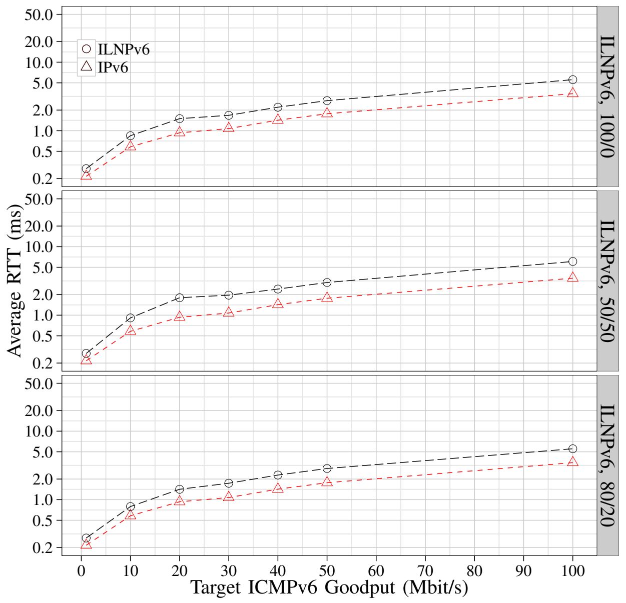
Locator selection can be optimised by eliminating redundant lookups in the node's FIB. Existing TCP/UDP implementations, including the BSD networking stack, may cache the FIB entry used to reach a socket's destination, and a similar enhancement could applied to ILNPv6 also.

Two events modify the remote Locator set for a session: DNS queries and ICMPv6 *Locator Update (LU)* messages [19]. LU messages are asynchronous signals from the remote correspondent node to indicate a change in its Locator set. The cache may be updated when either the Locator set or the forwarding entries change [35, Sec. 3.3.1 P2]. Thus,

[8]'Premature optimization is the root of all evil.' [34]



(a) 128 byte Ethernet frames



(b) 1514 byte Ethernet frames

Fig. 8. Mean RTT across ILNPv6 configurations (Error bars not visible due to scale)

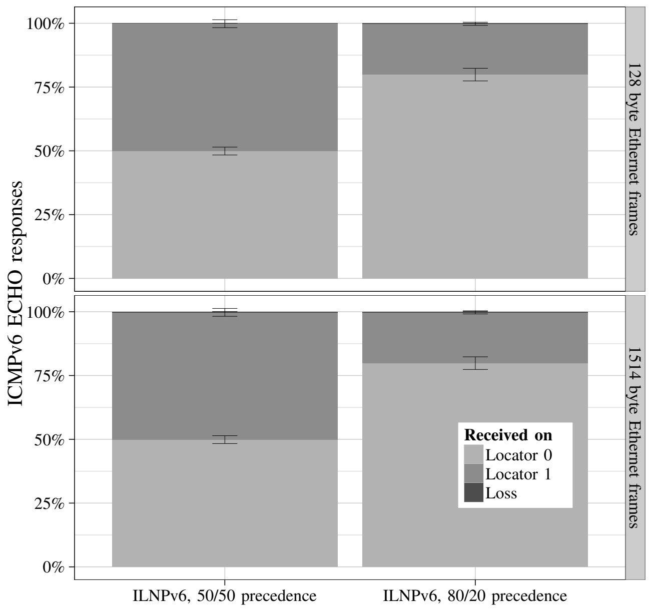


Fig. 9. Distribution of ICMPv6 ECHO responses received at a multihomed ILNPv6 host

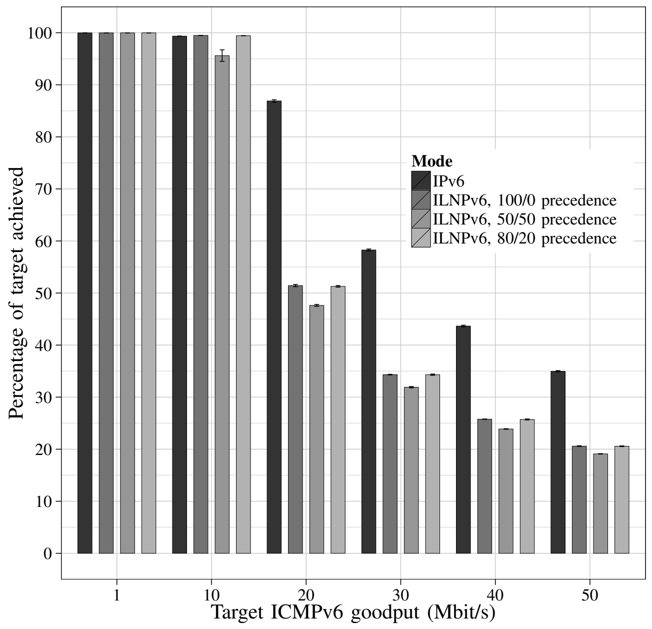


Fig. 10. Measured ICMPv6 goodput for 128 byte frames

$O(N)$ lookups for *each* packet may be substituted with $O(N)$ *advance* lookups. For an ILNPv6 host with a single default Locator, this cost becomes $O(1)$. Moreover, direct access to the FIB using `rtalloc()` [36] (not shown in Fig. 6) may require only 2 mutexes, improving concurrency control.

Our implementation uses the Nonce Option to differentiate ILNPv6 packets from IPv6 for session initiation. For convenience, we include this option in all ILNPv6 packets, but it should be possible to use a bit from the IPv6 *Flow Label*. This may ease demultiplexing in a dual-stack IPv6/ILNPv6 implementation, especially in site border routers, as they should not normally examine IPv6 Destination Options.

VII. Conclusion and Future Work

We have implemented ILNPv6 in FreeBSD to support host multihoming, showing it is possible to re-use IPv6 protocol code for ILNPv6. We evaluated relative performance using

ICMPv6 ECHO traffic. We found that the relative performance of the ILNPv6 implementation is similar to that for unmodified IPv6. We observe very close performance with large packets (1514 bytes), but degraded performance for small packets (128 bytes) above 10Mbps ($\approx$50% of IPv6). However, with the optimisations discussed in this paper, we expect the performance gap to narrow. So, we expect that host-based ID/Loc based on ILNP to have relatively small performance overhead at the end-systems.

The simple load balancing scheme described in Subsec. IV-H is intended to demonstrate how network control functions may be realised at hosts by application of the 'ID/Loc' split. Therefore, we do not examine interoperability with existing schemes load balancing schemes, e.g. site-based load balancing. Future work will extend the host-multihoming scenario to site-multihoming, by modifying the host implementation to combine with FreeBSD router functions. The address management mechanisms may be extended to support multi-path transport protocols, such as MP-TCP [9, Sec. 5.2].

Additionally, ILNP multi-homing could be integrated with mobility mechanisms based on ILNP [9, Sec. 6], leveraging new features such as network layer soft hand-off [37].

Acknowledgements

We are grateful to Cisco for their *University Research Programmme (URP)* in supporting this work.

References

[1] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984, IETF, Sep 2007.

[2] T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. *Computer Networks*, 45(1):45–54, 2004.

[3] X. Meng et al. IPv4 Address Allocation and the BGP Routing Table Evolution. *ACM Comp. Comm. Rev.*, 35:71–80, Jan 2005.

[4] L. Cittadini, W. Muhlbauer, S. Uhlig, R. Bush, P. Francois, and O. Maennel. Evolution of Internet Address Space Deaggregation: Myths and Reality. *IEEE JSAC*, 28(8):1238–1249, 2010.

[5] J. F. Shoch. A note on Inter-Network Naming, Addressing and Routing. IEN 19, IETF, Jan 1978.

[6] B. Carpenter, J. Crowcroft, and Y. Rekhter. IPv4 Address Behaviour Today. RFC 2101, IAB, Feb 1997.

[7] R. Atkinson, S. N. Bhatti, and S. Rose. DNS Resource Records for the Identifier-Locator Network Protocol (ILNP). RFC 6742, IRTF, Nov 2012.

[8] T. Li. Recommendation for a Routing Architecture. RFC 6115, IETF, Feb 2011.

[9] R. Atkinson and S. N. Bhatti. Identifier-Locator Network Protocol (ILNP) Architectural Description. RFC 6740, IRTF, Nov 2012.

[10] B. E. Carpenter, J. Crowcroft, and Y. Rekhter. IPv4 Address Behaviour Today. RFC 2101, IETF, Feb 1997.

[11] JP. Dionne et al. Gap Analysis for IPv4 Sunset. Internet-Draft draft-ietf-sunset4-gapanalysis-03, IETF, Jul 2013.

[12] R. Moskowitz et al. Host Identity Protocol. RFC 5201, IETF, Apr 2008.

[13] P. Nikander and J. Laganier. Host Identity Protocol (HIP) Domain Name System (DNS) Extensions. RFC 5205, IETF, Apr 2008.

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

[15] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972, IETF, Mar 2005.

[16] M. Bagnulo. Hash-Based Addresses (HBA). RFC 5535, IETF, Jun 2009.

[17] D. Meyer. The Locator Identifier Separation Protocol (LISP). *Internet Protocol Journal*, 11(1), Mar 2008.

[18] R. Atkinson and S. N. Bhatti. Identifier-Locator Network Protocol (ILNP) Engineering Considerations. RFC 6741, IRTF, Nov 2012.

[19] R. Atkinson and S. N. Bhatti. ICMP Locator Update Message for the Identifier-Locator Network Protocol for IPv6 (ILNPv6). RFC 6743, IRTF, Nov 2012.

[20] R. Atkinson and S. N. Bhatti. IPv6 Nonce Destination Option for the Identifier-Locator Network Protocol for IPv6 (ILNPv6). RFC 6744, IRTF, Nov 2012.

[21] R. Atkinson and S. N. Bhatti. ICMP Locator Update Message for the Identifier-Locator Network Protocol for IPv4 (ILNPv4). RFC 6745, IRTF, Nov 2012.

[22] R. Atkinson and S. N. Bhatti. IPv4 Options for the Identifier-Locator Network Protocol (ILNP). RFC 6746, IRTF, Nov 2012.

[23] R. Atkinson and S. N. Bhatti. Address Resolution Protocol (ARP) for the Identifier-Locator Network Protocol for IPv4 (ILNPv4). RFC 6747, IRTF, Nov 2012.

[24] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, IETF, Feb 2006.

[25] R. Draves et al. Default Address Selection for Internet Protocol version 6 (IPv6). RFC 6724, IETF, Sep 2012.

[26] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. R. Stevens. Basic Socket Interface Extensions for IPv6. RFC 3493, IETF, Mar 2003.

[27] R. Atkinson, S. Bhatti, and S. Hailes. ILNP: Mobility, Multi-homing, Localised Addressing and Security Through Naming. *Telecommunication Systems*, 42(3):273–291, Dec 2009.

[28] R. Atkinson, S. Bhatti, and S. Hailes. Evolving the Internet Architecture Through Naming. *IEEE JSAC*, 28(8):1319–1325, Oct 2010.

[29] The FreeBSD Project. sysctl(3). In *FreeBSD Library Functions Manual*. April 2010. http://www.freebsd.org/cgi/man.cgi?query=sysctl&sektion=3&manpath=FreeBSD+8.2-RELEASE.

[30] P. E. McKenney and K. F. Dove. Efficient Demultiplexing of Incoming TCP Packets. *ACM Comp. Comm. Rev.*, 22:269–279, 1992.

[31] R. Atkinson and S. N. Bhatti. Optional Advanced Deployment Scenarios for the Identifier-Locator Network Protocol (ILNP). RFC 6748, IRTF, Nov 2012.

[32] S. N. Bhatti and R. Atkinson. Secure & Agile Wide Area Virtual Machine Mobility. In *Proc. IEEE MILCOM 2011*, Oct 2012.

[33] R. Atkinson and S. Bhatti. Site-Controlled Secure Multi-homing and Traffic Engineering for IP. In *Proc. IEEE MILCOM 2009*, Oct 2009.

[34] Donald E. Knuth. Structured programming with go to statements. *Computing Surveys*, 6:261–301, 1974.

[35] G. Varghese. *Network Algorithmics*. Chapman & Hall/CRC, 2010.

[36] The FreeBSD Project. rtalloc(9). In *FreeBSD Kernel Developer's Manual*. December 2008. http://www.freebsd.org/cgi/man.cgi?query=rtalloc&sektion=9&manpath=FreeBSD+8.2-RELEASE.

[37] D. Phoomikiattisak and S. N. Bhatti. Network Layer Soft Handoff for IP Mobility. In *PM2HW2N 2013 - 8th ACM Intl. Wkshp. Performance Monitoring, Measurement and Evaluation of Heterogeneous Wireless and Wired Networks*, Nov 2013.