Network Working Group
Internet-Draft
Updates: 6740, 6741, 6748 (if approved)
Intended status: Experimental
Expires: 9 November 2026

S. N. Bhatti
University of St Andrews, UK
G. T. Haywood
Abertay University, UK
R. Yanagida
University of St Andrews, UK
R. W. Grimes
Independent, USA

8 May 2026

ILNP usage by IPv6 applications
draft-bhatti-ilnp-ip6-apps-00

Abstract

The Identifier Locator Network Protocol (ILNP) for IPv6 is described in Experimental RFCs 6740-6744. ILNP uses a different architecture to IPv6 but is implemented to work with IPv6. This document describes how unmodified IPv6 applications running on an ILNP-capable node can make use of ILNP. This document updates RFC6740, RFC6741, RFC6748.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-bhatti-ilnp-ip6-apps/.

Discussion of this document takes place on the Network Network Working Group mailing list (mailto:saleem@st-andrews.ac.uk).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 9 November 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction 3
1.1. Purpose 3
1.2. Rationale 4
2. Motivation 4
3. Conventions and Definitions 5
3.1. Definitions from other documents 5
4. Examples used in this document 6
4.1. Use of the sockets API 6
5. Practical addressing and naming for ILNP 6
5.1. Name resolution process 7
5.2. Name resolution for an application 8
5.3. DNS queries for L64 and NID RRs 9
6. IPv6 server application using ILNP 10
6.1. Case 0: "Wildcard" addressing 10
6.2. Case 1: Predefined I-LV values 11
6.3. Case 2: Predefined L64 and NID values 12
6.4. Server-side use of /etc/hosts 13
6.5. Server-side exceptions 14
7. IPv6 client application using ILNP 14
7.1. Client-side use of /etc/hosts 14
7.2. Client-side exceptions 15
8. Communication session management 15
9. Security Considerations 16
10. Privacy Considerations 16
11. IANA Considerations 16
12. References 16
12.1. Normative References 16
12.2. Informative References 18
Acknowledgements 19
Authors' Addresses 19

1. Introduction

ILNP is designed to be backwards compatible "on-the-wire" with IPv6, and to be incrementally deployable such that only nodes that need to use ILNP need to be modified, typically end-systems only. No modifications are required to switches or routers. Additionally, ILNP is designed to be backwards compatible with _IPv6 applications_. For many IPv6 applications, there should be no requirements for re-engineering or re-compilation of existing program binaries: existing IPv6 programs should be able to run on a node that has been updated with an ILNP-capable network stack.

When an IPv6 application uses ILNP communication, it is important that such usage is:

* _intentional_: comes from explicit actions taken to use ILNP, e.g. there is a clear choice about the use of ILNP communication for IPv6 applications from systems configuration; and

* _predictable_: avoids surprises, as much as is possible, from the use of ILNP, e.g. that ILNP communication does not result in the IPv6 application entering unsupported states or undesirable modes of operation.

However, even with carefully planned and intentional usage, the wide diversity of applications means that it is possible that the behaviour of some applications is not predictable.

ILNP is defined for use with IPv6 and for use with IPv4, but all references in this document to ILNP are for IPv6 only.

1.1. Purpose

The purpose of this document is to describe how unmodified IPv6 applications can use ILNP-based communication:

1. The mechanisms by which ILNP communication is possible for unmodified IPv6 applications.

2. How unmodified IPv6 applications can run on an ILNP-capable node.

3. Which IPv6 applications should work with ILNP, and which are unlikely to work.

The mechanism by which an ILNP-capable node communicates with a non-ILNP capable node is to fall back to using IPv6, as described in Section 8 of [RFC6740], Section 10 of [RFC6741], Section 5 of [RFC6743], and Section 6 of [RFC6744].

This document discusses only communication based on unicast addressing: multicast communication for ILNP is as for IPv6.

1.2. Rationale

It is expected that new _ILNP-aware_ applications will in the future make use of access to the new addressing data-types in ILNP to have more flexible connectivity and control of traffic flows. This will depend on the availability of suitable a API for access to ILNP-specific addressing information.

In parallel, existing _IPv6 applications_ should be able to make use of ILNP because it was designed to be backwards compatible with IPv6. Indeed, it would be beneficial if as many as possible of the benefits of ILNP could be used by _unmodified_ IPv6 applications. So, this document describes how existing IPv6 applications can use ILNP without being modified.

2. Motivation

Why should an IPv6 application use ILNP?

A complete answer to this question, and any detailed discussion, is beyond the scope of this document.

As a short answer to the question above, the different approach to addressing and connectivity in ILNP means that, for example, an existing, unmodified IPv6 application could gain in the following ways:

* Improved privacy and security at the packet level [BHY2021] [HB2024] [HB2025].

* Combined mobility-multihoming capability under end-system control [YB2024].

* Provision of multipath connectivity for existing transport protocols [YB2019].

This additional control for connectivity and communication for an application with ILNP is designed using an end-to-end approach, allowing all the functionality to work together harmoniously. Also, the end-to-end approach means the functionality can be implemented without requiring the use of proxies, tunnelling, or address translation mechanisms.

Complete solutions for unmodified IPv6 applications to use ILNP in this way are being developed and tested at the time of writing. Meanwhile, the work cited above, using results from ILNP implementations in FreeBSD and Linux, with ongoing testing and experiments over international connectivity at IETF meetings and IETF Hackathon events, show that there are realistic possibilities for ILNP to operate across global IPv6 connectivity, and that many existing IPv6 applications could work unmodified over ILNP.

3. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3.1. Definitions from other documents

The following terms are defined in [RFC6740]:

* Locator, L64

* Node Identifier, NID

* Identifier-Locator Vector, I-LV

* I-L Communications Cache, ILCC

The following term is defined in [draft-bhatti-ilnp-preference]:

* ordered I-LV sequence, {A_a}

The notation for textual representations for I-LV values is defined in [draft-bhatti-ilnp-textual-representations].

The reference to the C sockets API extensions for IPv6 is in relation to [RFC3493], particularly the use of:

* the struct addrinfo data structure.

* the struct sockaddr_in6 data structure.

* the getaddrinfo(3) C library function call.

4. Examples used in this document

In order to describe the interactions for communication, a client-server mode of operation is assumed in the explanations and descriptions. However, the approach is general: the ILNP client is a communication initiator, the ILNP server is a communication responder. The communication process is described in terms of:

1. An ILNP-capable node waiting to accept incoming communication requests from a remote ILNP-capable node, e.g. incoming TCP connections or UDP sessions. For the purposes of this document, such a node is called an _ILNP server node_.

2. An ILNP-capable node wishing to initiate communication to a remote ILNP-capable node. For the purposes of this document, such a node is called an _ILNP client node_.

So, for our purposes, an IPv6 server application is running on an ILNP server node, and an IPv6 client application is running on an ILNP client node.

4.1. Use of the sockets API

This document makes use of the sockets API and related family of function calls, data structures, data-types, and value definitions in C (which are also defined as part of POSIX.1-2008 / POSIX.1-2017 / POSIX-1.2024).

In the sockets API, _address family_ definitions exist for use with the function calls and data structures to identify the type of addressing being used: AF_INET for IPv4, AF_INET6 for IPv6 [RFC3493]. In the future, new addressing extensions for sockets might be defined for ILNP, just as extensions were defined for IPv6 in addition to IPv4. For now, this document describes how ILNP is used for backwards compatibility with IPv6 applications only, with any ILNP I-LV values presented to applications as type AF_INET6 so that I-LV values can be used as IPv6 addresses.

5. Practical addressing and naming for ILNP

The key mechanisms by which unmodified IPv6 applications can make use of ILNP is through the syntactic compatibility in naming and addressing formats between ILNP and IPv6. For name resolution, ILNP uses the same sources as for IPv6, e.g. entries in /etc/hosts or responses from DNS. I-LV values are 128-bit values that are syntactically equivalent to an IPv6 address. So, for carrying addressing information across the network, an ILNP packet uses the address fields in the IPv6 packet header to carry I-LV values.

As ILNP packets are "on-the-wire" identical to IPv6 packets, network components, such as routers and switches, can forward ILNP packets, and I-LV values can be handled like IPv6 addresses. However, ILNP operates end-to-end, so ILNP-aware nodes can act upon the semantics of L64 and NID values end-to-end to enable the various functionality that is possible for ILNP, as described in [RFC6740] and [RFC6741].

5.1. Name resolution process

It is possible for an ILNP-capable node to determine if a remote node is also ILNP-capable through name resolution using existing mechanisms, and so initiate ILNP communication.

When a name, such as a fully-qualified domain name (FQDN), is passed by an application to a node, name resolution is performed using /etc/hosts or DNS. The resolution of a name to an address selects the type of communication -- IPv4 or IPv6 -- that is possible and could be initiated by an application. The type information is part of the name resolution, the most common examples being:

* the implicit type information in the textual representation of an address value in /etc/hosts:
  - decimal-dot notation for IPv4, such as "123.12.34.45".
  - hexadecimal-colon notation for IPv6, such as "2001:db8:12:34::abcd:1".
* the explicit type value of a DNS Resource Record (RR) returned in a DNS response:
  - type A for IPv4.
  - type AAAA for IPv6.

So, similarly, for ILNP we have type information included with addressing values:

* notations for the textual representations of L64, NID, and I-LV values for use in /etc/hosts as defined in [draft-bhatti-ilnp-textual-representations], such as "2001-db8-12-34.0+0+abcd+1" for an I-LV.
* DNS RRs for L64 and NID values, with type values assigned by IANA, as documented in [RFC6742].

Although it is expected that normal operation for name resolution will be using DNS, the use of /etc/hosts can be a convenient method in some cases, e.g. for experimentation and debugging, especially on a local testbed. Some guidance on the use of /etc/hosts is provided in Section 6.4 and Section 7.1. Also, server systems especially might need application-specific addressing configuration, and that is out of scope for this document.

5.2. Name resolution for an application

The socket(2) API uses the getaddrinfo(3) library function to resolve names and provide for IPv6 applications a list of 128-bit IPv6 address in an instance of a struct addrinfo list. getaddrinfo(3) has the function prototype:

```
int getaddrinfo(const char *nodename, const char *servname,
                const struct addrinfo *hints, struct addrinfo **res);
```

The typical code pattern for name resolution is:

1. The ai_family member of hints is set to AF_UNSPEC (or AF_INET6 for IPv6 only) before invoking getaddrinfo(3) with a value for nodename to resolve. (servname might also be used but is not relevant for this document.)

2. Upon successful return, res holds a struct addrinfo list with a sequence of IPv6 address values that could be used by the application.

3. A value from res is copied into a struct sockaddr_in6 instance for use in socket(2).

For an ILNP-capable node, to allow unmodified IPv6 applications to use ILNP, the behaviour of getaddrinfo(3) MUST be:

1. upon invocation:

   * if the nodename provided matches I-LV entries from /etc/hosts, then an _ordered I-LV sequence_, {A_a}, MUST be constructed from the matching entries;

   * otherwise nodename is used in DNS queries so that L64 and NID RRs can be retrieved (as well as AAAA and A RRs, if defined, for backwards compatibility), and the returned L64 and NID RRs MUST be used to construct an _ordered I-LV sequence_, {A_a}.

2. upon successful return for the API user, the list in res MUST contain at its _start_ the list of the I-LV values from {A_a} with the struct addrinfo instances marked as AF_INET6 as if they are IPv6 addresses.

3. below the API, the ILCC and internal state for the ILNP node MUST record that the name resolution did include I-LV values, so that a subsequent socket(2) invocation with a struct addrinfo_in6 instance containing an I-LV value from res will return a descriptor to a socket that will be an ILNP communication end-point.

The _ordered I-LV sequence_, {A_a}, contains I-LV values in a preferred ordering for the remote node. The process for constructing {A_a} is described in [draft-bhatti-ilnp-preference], but that process is not visible to the user.

So, the socket(2) family of calls using res will have the correct type information for an IPv6 application, and API calls will also result in an ILNP communication end-point. That is, the application will have a reference to an IPv6 communication end-point (the socket descriptor that is returned), but the type of the socket that will be created will be for ILNP communication. Hence, an unmodified IPv6 application will be able to engage in ILNP-based communication.

5.3. DNS queries for L64 and NID RRs

A simple way to implement a DNS query to select L64, NID, AAAA, and A RRs that match the same nodename is to use DNS query type of ANY, i.e. QTYPE=ANY. While a DNS query with QTYPE=ANY is still possible for deployed DNS resolvers, its use is subject to variable and non-consistent behaviour in responses, as described in [RFC8482]. This non-consistent response behaviour could be due to a number of reasons, e.g. to avoid traffic amplification attacks [RFC5358] when using UDP for DNS queries, or due to local policy.

Alternatively, parallel DNS queries with different QTYPE values could be made by an implementation of getaddrinfo(3), e.g. multiple queries with QTYPE set, in turn, to L64, NID, AAAA, and A. However, L64 and NID RRs might not be returned first -- one of the other parallel queries could be answered first and that could be enough for the call to getaddrinfo(3) to complete successfully.

As the ordering, delay, and content of DNS responses to QTYPE=ANY or to multiple parallel queries are not well-defined, res might not contain I-LV values, even if L64 and NID RRs are defined for nodename in the DNS. This means that an I-LV might not always be selected after a call to getaddrinfo(3) when DNS is used.

Even if I-LV values are included in res, with the added recommendation for "Happy Eyeballs" [RFC8305], an ILNP communication might not always be initiated between IPv6 applications running on ILNP-capable nodes. This is not an issue particular to ILNP: the same is true for IPv6 and IPv4, i.e. the combination of the non-consistent DNS responder behaviour and the "Happy Eyeballs" recommendation could mean that IPv4 is selected for communication between IPv6-capable applications both running on IPv6-capable nodes.

6. IPv6 server application using ILNP

For an ILNP server node, the IPv6 server application needs to establish a communication end-point that can accept incoming ILNP communication requests.

An ILNP server node, or individual IPv6 application, might be invoked via local or application-specific configuration, and be initialised with a communication end-point to accept incoming communication requests in the following ways:

* Case 0: Use of "wildcard" addressing, i.e. no ILNP-specific information is configured when the IPv6 server application is invoked.

* Case 1: Predefined I-LV values are used directly, e.g. the ILNP server node has defined for it one or more specific whole I-LV values (or names that resolve to whole I-LV values) in place of IPv6 addresses in the application-specific configuration.

* Case 2: Separate L64 and NID values, e.g. the ILNP server node is configured with multiple L64 and NID values and so can form I-LV values, which can then be used by the IPv6 application as if the I-LV values are IPv6 addresses.

6.1. Case 0: "Wildcard" addressing

It is NOT RECOMMENDED that IPv6 server applications make use of ILNP communication as described in this sub-section. The usage of ILNP will not be intentional and is unlikely to be predictable.

An IPv6 server application code pattern might typically invoke bind(2) using IN6ADDR_ANY_INIT for IPv6, so that it can accept incoming connections for any and all IPv6 addresses that it currently has configured, IPv4 and / or IPv6.

For an IPv6 server application running on an ILNP server node, it is RECOMMENDED that the server application is invoked using a locally defined I-LV (or name resolving to an I-LV) -- please see

Section 6.2. This could be achieved through local configuration files or command-line arguments for invocation of the server application.

6.2. Case 1: Predefined I-LV values

It is RECOMMENDED that an IPv6 server application running on an ILNP server node is configured and invoked as described in this sub-section. This usage is intentional and should result in predictable behaviour. For practical purposes, it is also the easiest and clearest method of enabling unmodified IPv6 server applications to make use of ILNP communication.

The scenario described here is for the case when I-LV values are to be used directly in place of IPv6 addresses:

1. It is RECOMMENDED that predefined I-LV values, or names resolving to such I-LV values, should be configured for the ILNP server node.

2. It is RECOMMENDED that these predefined I-LV values, or names resolving to such I-LV values, should be used when starting an IPv6 application to explicitly make use of ILNP-based communication.

As an example, predefined I-LV values could be configured as described in Section 6.4.

When a predefined I-LV, or a name resolution resulting in a predefined I-LV, is used in the invocation to bind(2), the ILNP server node:

1. SHOULD allow ILNP communication for that I-LV at the time of invocation.

2. SHOULD maintain ILNP communication for that I-LV value after invocation, consistent with the validity of the L64 value for the I-LV.

The "SHOULD" in points 1 and 2 above allows flexibility for application-specific behaviour or local policy. Also, it allows flexibility for whether or not such communication end-points are to be made discoverable to remote client applications for ILNP communication, e.g. by a Dynamic Secure DNS update [RFC3007] that creates or updates L64 and NID RRs for that I-LV through some out-of-application mechanism.

For point 2 above, it is possible that a L64 value becomes unavailable and then becomes available again, e.g. due to presence (or not) of IPv6 prefix values in IPv6 Routing Advertisements (RAs) coupled with the behaviour of ILNP dynamic multihoming and mobility capability [YB2019] [YB2024]. The "SHOULD" in point 2 gives flexibility for application-specific behaviour or local policy in this respect. If the behaviour is implemented and utilised, the IPv6 application gains connectivity capability it would never have had. If the behaviour is not implemented or utilised, nothing has been lost with respect to the normal expected operation of the IPv6 server application, but client applications would still gain from using ILNP communication -- please see Section 2. However, when the behaviour is implemented, and the I-LV lifetime expires, a previously bound socket becomes invalid, and the behaviour for such a situation is system-specific.

6.3. Case 2: Predefined L64 and NID values

It is NOT RECOMMENDED that an IPv6 server application running on an ILNP server node is configured and invoked as described in this sub-section. Although the usage is intentional, it might result in behaviour that is not predictable.

The scenario described here is a possibility. However, it is not expected that unmodified IPv6 server applications will be invoked in such a configuration.

An ILNP server node might have defined for it multiple L64 and NID values from which it can form I-LV values as described in [draft-bhatti-ilnp-preference]. These could be configured locally or learned dynamically from DNS. L64 and NID values from DNS RRs have a Time-To-Live (TTL) value, but it is possible that local mechanisms also allow L64 and NID values to have limited lifetimes much like the DNS TTL. If separate L64 and NID values are defined without specific lifetimes, then the behaviour will be as for predefined I-LV values as in Section 6.2.

It is possible that, for example, predefined NID values are configured locally with no expiry time, while L64 values are discovered dynamically (typically as address prefix values from IPv6 RAs) and might have a limited lifetime. In this case I-LV values could be transient because of the L64 lifetime. If a NID value is defined without a specific lifetime, and the L64 value is learned dynamically (e.g. an address prefix from an IPv6 RA) but has a "long" lifetime (e.g. a day, or a week), this behaviour is similar to the use of SLAAC [RFC4862], and so, with care, the scenario of Section 6.2 could still be employed for an ILNP server node.

However, if the L64 and NID values each have limited lifetimes, then this impacts the overall lifetimes of the I-LVs that are formed from those L64 and NID values. The IPv6 application will, effectively, be using addresses that will "expire" to create sockets on which to accept incoming communication sessions.

When the invocation of bind(2) is made with a specific I-LV formed from the separate L64 and NID values (as described in [draft-bhatti-ilnp-preference]), the ILNP server node:

1. SHOULD allow ILNP communication for that I-LV at the time of invocation; and

2. SHOULD maintain ILNP communication for that I-LV consistent with the validity of those L64 and NID values that constitute that I-LV.

The "SHOULD" in point 1 and 2 above allows flexibility for application-specific behaviour or local policy. Also, it allows flexibility for whether or not such communication end-points are to be made discoverable to remote client applications for ILNP communictaion, e.g. by a Dynamic Secure DNS update [RFC3007] that creates or updates to L64 and NID RRs through some out-of-application mechanism. However, when the behaviour is implemented, and the I-LV lifetime expires, a previously bound socket becomes invalid, and the behaviour for such a situation is system-specific.

6.4. Server-side use of /etc/hosts

A name could be defined for lookup of an I-LV before bind(2) is invoked. This could be in DNS, as a client application would also need that information to initiate communication. However, a name could (also) be a local name defined in /etc/hosts for an I-LV that the IPv6 server application is configured to use, the local name being provided via command-line arguments or application-specific configuration files. Care should be taken if I-LV entries are used in /etc/hosts with corresponding L64 and NID entries in DNS: the latter will be required for client applications under normal operation. Examples of names and I-LV entries in /etc/hosts are given in Figure 1. This could be for two different servers on the same physical / virtual node, or some other arrangement, but that choice is left to the user.

```
2001-db8-0-1.abcd+0+a+1     server-a1.local  # a local I-LV
2001-db8-0-1.abcd+0+a+2     server-a2.local  # another local I-LV
```

Figure 1: Example entries in the ‘/etc/hosts‘ file for I-LV values for ILNP server nodes.

This does not preclude any application-specific configuration or local policy control for I-LV selection.

6.5. Server-side exceptions

With use of ILNP communication for IPv6 applications, care is needed in choosing I-LV values for initial configuration and instantiation. There are situations where ILNP communication cannot be initiated transparently for an IPv6 server. In most cases, this is likely to be where the server application (or accompanying client application) uses IPv6 address values directly, e.g. within the code base or in configuration files. For example, if a server application expects to bind(2) to a "hard-coded" address, or to use a specific interface on a node, or if a server application manipulates address bits directly in user-space as part of its behaviour. This could be true especially for some control-plane and management-plane applications.

7. IPv6 client application using ILNP

Typically, an IPv6 client application will invoke getaddrinfo(3) with a nodename for a remote node, then use a value from the res list, often the first value in the res list, to make a call to socket(2), and then start a communication session. Using the mechanism described in Section 5.2, the first value in res will be for an I-LV. There might also be other I-LVs in res, but it is not possible for the IPv6 application to know that. A subsequent call, e.g. to connect(2) with that first value from res will create an ILNP communication end-point.

7.1. Client-side use of /etc/hosts

If DNS is not configured or if the use of /etc/hosts is desired, examples of names and I-LVs for ILNP server nodes that could be used by the IPv6 client application are given in Figure 2. Care should be taken if I-LV entries are used in /etc/hosts with corresponding L64 and NID entries in DNS: it is RECOMMENDED that client systems use DNS under normal operation.

```
2001-db8-0-1.abcd+0+a+1     server-a1.ilnp  # a remote server
2001-db8-0-1.abcd+0+a+2     server-a2.ilnp  # another remote server
```

Figure 2: Example entries in the ‘/etc/hosts‘ for use by a client to lookup I-LV values for a remote ILNP server node.

This does not preclude any application-specific configuration or local policy control for I-LV selection.

7.2. Client-side exceptions

In most cases, client applications initiate communication after a name resolution is performed to find addressing information and initiate a communication end-point, as described in Section 5.2. Then, the client application typically only uses a reference to the communication end-point (a socket descriptor), and is usually not concerned with the exact values in the addressing information. So, it is expected that most IPv6 client applications will be able to make use of ILNP-based communication.

However, after using getaddrinfo(3), if the client application chooses a value from the res list that is not an I-LV, then ILNP communication will not be initiated.

Also, as is true for the server-side, as described in Section 6.5, if the client application (or accompanying server application) uses IPv6 address values directly, e.g. within the code base or in configuration files, ILNP communication might not work correctly. Again, this could be true especially for some control-plane and management-plane applications.

Additionally, as already discussed in Section 5.3, the effects of the DNS responder behaviour coupled with the "Happy Eyeballs" recommendations might result in ILNP not being initiated by a client application even though it is possible.

8. Communication session management

Once a communication session is successfully initiated (e.g. a TCP connection), the handling of addressing for each end-point can be summarised as:

1. There is a Source I-LV and a Destination I-LV, consisting, respectively of L64_s and NID_s, and L64_d and NID_d.

2. NID_s and NID_d values MUST remain constant for the duration of the session.

3. The Source node or Destination node MAY signal changes, respectively, to their L64_s or L64_d values by the use of Locator Update (LU) message [RFC6743].

Please note that this is a summary only, full details of the state management process for L64 and NID values are given in [RFC6740] [RFC6741].

However, the user-level control of the session via the sockets API is unchanged for the IPv6 application.

9. Security Considerations

There are no new security considerations.

The existing security mechanisms already defined for ILNP remain unchanged (please see Section 9 of [RFC6740], Section 11 of [RFC6741]). The use of the ILNP Nonce Header [RFC6744] would offer extra lightweight protection for communication flows at the IP packet level, compared to that which is available for IPv6 without the use of IPsec.

10. Privacy Considerations

There are no new privacy considerations.

The existing identity privacy and location privacy properties already defined for ILNP remain unchanged (please see Section 10 of [RFC6740], Section 12 of [RFC6741], Section 7 of [RFC6748]). NID values can also be generated using privacy mechanisms such as [RFC8981]. ILNP can also use enhanced identity privacy and location privacy mechanisms which remain backwards compatible with IPv6, such as described in [BHY2021] [HB2024] [HB2025].

11. IANA Considerations

This document has no IANA actions.

12. References

12.1. Normative References

[draft-bhatti-ilnp-preference]
Bhatti, S. N., "ILNP addressing using Preference values", 8 May 2026. A related draft that is being produced in parallel.

[draft-bhatti-ilnp-textual-representations]
Bhatti, S. N., Haywood, G. T., Yanagida, R., and R. W. Grimes, "ILNP textual representations", 8 May 2026. A related draft that is being produced in parallel.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3007] Wellington, B., "Secure Domain Name System (DNS) Dynamic Update", RFC 3007, DOI 10.17487/RFC3007, November 2000, <https://www.rfc-editor.org/rfc/rfc3007>.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, DOI 10.17487/RFC3493, February 2003, <https://www.rfc-editor.org/rfc/rfc3493>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/rfc/rfc4862>.

[RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, DOI 10.17487/RFC5358, October 2008, <https://www.rfc-editor.org/rfc/rfc5358>.

[RFC6740] Atkinson, RJ. and SN. Bhatti, "Identifier-Locator Network Protocol (ILNP) Architectural Description", RFC 6740, DOI 10.17487/RFC6740, November 2012, <https://www.rfc-editor.org/rfc/rfc6740>.

[RFC6741] Atkinson, RJ. and SN. Bhatti, "Identifier-Locator Network Protocol (ILNP) Engineering Considerations", RFC 6741, DOI 10.17487/RFC6741, November 2012, <https://www.rfc-editor.org/rfc/rfc6741>.

[RFC6742] Atkinson, RJ., Bhatti, SN., and S. Rose, "DNS Resource Records for the Identifier-Locator Network Protocol (ILNP)", RFC 6742, DOI 10.17487/RFC6742, November 2012, <https://www.rfc-editor.org/rfc/rfc6742>.

[RFC6743] Atkinson, RJ. and SN. Bhatti, "ICMP Locator Update Message for the Identifier-Locator Network Protocol for IPv6 (ILNPv6)", RFC 6743, DOI 10.17487/RFC6743, November 2012, <https://www.rfc-editor.org/rfc/rfc6743>.

[RFC6744] Atkinson, RJ. and SN. Bhatti, "IPv6 Nonce Destination Option for the Identifier-Locator Network Protocol for IPv6 (ILNPv6)", RFC 6744, DOI 10.17487/RFC6744, November 2012, <https://www.rfc-editor.org/rfc/rfc6744>.

[RFC6748] Atkinson, RJ. and SN. Bhatti, "Optional Advanced Deployment Scenarios for the Identifier-Locator Network Protocol (ILNP)", RFC 6748, DOI 10.17487/RFC6748, November 2012, <https://www.rfc-editor.org/rfc/rfc6748>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/rfc/rfc8305>.

[RFC8482] Abley, J., Gudmundsson, O., Majkowski, M., and E. Hunt, "Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY", RFC 8482, DOI 10.17487/RFC8482, January 2019, <https://www.rfc-editor.org/rfc/rfc8482>.

[RFC8981] Gont, F., Krishnan, S., Narten, T., and R. Draves, "Temporary Address Extensions for Stateless Address Autoconfiguration in IPv6", RFC 8981, DOI 10.17487/RFC8981, February 2021, <https://www.rfc-editor.org/rfc/rfc8981>.

12.2. Informative References

[BHY2021] Bhatti, S., Haywood, G., and R. Yanagida, "End-to-End Privacy for Identity & Location with IP", IEEE, 2021 IEEE 29th International Conference on Network Protocols (ICNP) pp. 1-6, DOI 10.1109/icnp52444.2021.9651909, November 2021, <https://doi.org/10.1109/icnp52444.2021.9651909>.

[HB2024] Haywood, G. and S. Bhatti, "Defence against Side-Channel Attacks for Encrypted Network Communication Using Multiple Paths", MDPI AG, Cryptography vol. 8, no. 2, pp. 22, DOI 10.3390/cryptography8020022, May 2024, <https://doi.org/10.3390/cryptography8020022>.

[HB2025] Haywood, G. and S. Bhatti, "Ephemeral Node Identifiers for Enhanced Flow Privacy", MDPI AG, Future Internet vol. 17, no. 5, pp. 196, DOI 10.3390/fi17050196, April 2025, <https://doi.org/10.3390/fi17050196>.

[YB2019] Yanagida, R. and S. Bhatti, "Seamless internet connectivity for ubiquitous communication", ACM, Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers pp. 1022-1033, DOI 10.1145/3341162.3349315, September 2019, <https://doi.org/10.1145/3341162.3349315>.

[YB2024] Yanagida, R. and S. Bhatti, "Mobility窶ｦultihoming Duality", MDPI AG, Future Internet vol. 16, no. 10, pp. 358, DOI 10.3390/fi16100358, October 2024, <https://doi.org/10.3390/fi16100358>.

Acknowledgements

The authors are grateful to the many members of the IETF community for their feedback on ILNP during IETF meetings, and to the IETF NOC Team who made possible testing and experiments for ILNP during those meetings and the IETF Hackathon events.

Alistair Woodman and _NetDEF_ supported G.T. Haywood in an internship for initiating a FreeBSD implementation of ILNP for his PhD studies.

Time Warner Cable partly supported R. Yanagida for a Linux implementation of ILNP during his PhD studies.

This work was partly supported by the _ICANN Grant Program_.

Authors' Addresses

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

Gregor T. Haywood
Abertay University, UK
Email: g.haywood@abertay.ac.uk

Ryo Yanagida
University of St Andrews, UK
Email: ryo@htonl.net

Rodney W. Grimes
Independent, USA
Email: rgrimes@FreeBSD.org