How to use IPv6 Link-Local Addresses in Applications
draft-smith-ipv6-link-locals-apps-00

Abstract
IPv6 Link-Local addresses can be used by applications. Doing so when possible will provide robustness and security benefits to the application. This memo describes the properties of Link-Local addresses and the benefits and limitations of using them in applications. It then describes how to use them in applications that use the Sockets API.

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 http://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 April 21, 2016.

Copyright Notice
Copyright (c) 2015 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 (http://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. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.  Introduction ........................................... 2
2.  Link-Local Prefix and Address Attributes ................. 2
3.  Benefits ................................................. 3
4.  Limitations ............................................ 4
5.  Using Link-Locals with the Sockets API ................. 5
   5.1.  sin6_scope_id ..................................... 5
   5.2.  getaddrinfo() ..................................... 6
6.  Security Considerations .................................... 7
7.  Acknowledgements ....................................... 7
8.  Change Log [RFC Editor please remove] .................... 7
9.  Informative References ................................... 7
Author's Address ........................................... 9

1.  Introduction

IPv6 Link-Local addresses can be used by applications. Doing so when possible will provide robustness and security benefits to the application. This memo describes the properties of Link-Local addresses and the benefits and limitations of using them in applications. It then describes how to use them in applications that use the Sockets API.

2.  Link-Local Prefix and Address Attributes

   o The Link-Local unicast prefix is fe80::/10 [RFC4291].

   o All IPv6 interfaces are required to have at least one Link-Local unicast address [RFC4291].

   o Link-Local addresses have link-local scope, uniquely identifying interfaces only within a single link [RFC4007].

   o Across multiple links attached to the same node there may be duplicate Link-Local addresses. These duplicate Link-Local addresses are disambiguated using scope zone information, or more simply, zone information, which describes on which specific link the chosen Link-Local address exists [RFC4007].

   o Link-Local addresses have infinite preferred and valid lifetimes, meaning they never expire or age out [RFC4862].

   o A specific Link-Local address is constantly present on an interface (due to its infinite preferred and valid lifetime)
unless actively replaced by an event such as the interface being
reinitialised or the interface being attached to a different link
(e.g., attached to a different wireless network access point)
[RFC4862].

- The Link-Local prefix is therefore present on all IPv6 enabled
  links [RFC4291], regardless of the presence of a router and router
  advertisements.

- By default, a host treats the Link-Local prefix as on-link
  [RFC5942].

- Hosts use IPv6 Stateless Address Autoconfiguration (SLAAC)
  [RFC4862] to form a Link-Local address when an interface is
  enabled [RFC6434].

- Routers must be able to use IPv6 SLAAC [RFC4862] to form a Link-
  Local address when an interface is enabled [RFC6434].

- A Link-Local unicast address is formed by combining fe80::/10 with
  a link-specific format interface identifier (IID), positioned in
  the lower part of the IPv6 address [RFC4862].

- As most IIDs are 64 bits in length [RFC4291][RFC7421], the Link-
  Local prefix on a link is most commonly fe80::/64.

- Link-Local packets can be forwarded by routers, but only back onto
  the same link they came from [RFC4007][RFC4291].

- By default, during default address selection, Link-Local addresses
  are preferred as IPv6 source and/or destination addresses over all
  other address types, with exception to the loopback address
  [RFC6724].

- The Link-Local prefix is not advertised in IPv6 Router
  Advertisement Prefix Information Options [RFC4861].

3. Benefits

There are a number of benefits to using Link-Local addresses in
applications.

- As all interfaces are required to have Link-Local addresses and
  therefore the Link-Local prefix is always present on a link,
  application connectivity is independent of the presence of a
  router. This means that connectivity can be established without
the presence of a router, or will not be impacted by the failure of a router, unless the router is part of the forwarding path between the Link-Local address application end-points.

- As Link-Local addresses have infinite preferred and valid lifetimes, and persist on an interface unless actively replaced, including across transient link failures, application connections will persist for as long as the application needs them to unless a period of packet loss occurs which the application’s transport protocol cannot tolerate. This is in contrast to the use of Unique Local Addresses (ULA) [RFC4193] or Globally Unique Addresses (GUA) [RFC4291] for application end-points, whose lifetimes may expire if not refreshed, causing the application’s connection end-points to become invalid and therefore unavailable, despite the link continuing to provide connectivity between the involved hosts.

- Assuming an application only uses Link-Local addresses, due to Link-Local traffic being limited to a single link, the potential set of application attackers is limited to the set of nodes attached to the same link. This set of link-attached nodes will likely be significantly smaller than sets of nodes with either ULA or GUA addresses that could reach the application from off-link locations. The likely close geographic proximity of nodes attached to the same link would also provide a disincentive to attack applications that are only using Link-Local addresses.

4. Limitations

There are also a number of limitations to using Link-Local addresses in applications.

- Link-Local addresses cannot be used for application connections that span multiple links.

- On some types of links, such as "Private VLANs" [RFC5517], a host may not be able to reach another host’s Link-Local address. This is due to the hub-and-spoke forwarding constraints imposed by the link on some nodes, and the default on-link assumption for the Link-Local prefix [RFC5942]. An application could try to overcome this limitation by attempting to connect to the target host’s ULA or GUA addresses if known, as full reachability might be available for these address types over these types of networks. [I-D.smith-6man-link-locals-off-link] proposes a method which would restore Link-Local address reachability over these types of links, while retaining hub-and-spoke forwarding constraints at the link-layer.
A Link-Local address is ambiguous, as by itself it does not inherently identify the link it is a member of, unlike ULA or GUA addresses. Consequently, applications have to be prepared to handle Link-Local zone information, which specifies on which link a Link-Local address is present. This may include providing user interface elements to allow an application user to specify per Link-Local address zone information.

When providing a Link-Local address to an application peer for use as a referral, it is best to ensure that the application peer is a member of the same link. This should then ensure that the application peer receives both the Link-Local address and correct zone information, implicit from the ingress interface the application peer received the referral over. For example, Multicast DNS achieves this by using link-local scope multicast destination addresses [RFC6762]. A Link-Local address should not be supplied to an off-link application peer for use as a referral unless the application knows the peer will have the correct zone information for the address, or can cope successfully with Link-Local address connection failure or securely with successful connection to an incorrect node (e.g., via some form of secure node or application level authentication).

5. Using Link-Locals with the Sockets API

5.1. sin6_scope_id

The ‘sockaddr_in6’ structure is used by the Sockets API [RFC3493] to hold IPv6 socket address data.

Within the ‘sockaddr_in6’ structure, the unsigned 32 bit ‘sin6_scope_id’ integer field identifies the scope or zone for the IPv6 address carried in the ‘sin6_addr’ sub-structure.

In the case of Link-Local addresses, the ‘sin6_scope_id’ field identifies the interface where the Link-Local address is located.

In the Sockets API, interfaces are identified using an unsigned integer, known as an ‘interface index’. The if_nametoindex() and if_indextoname() functions [RFC3493] are used to convert from an interface name to an interface index and vice-versa.

When using Link-Local addresses in a ‘sockaddr_in6’ structure, the ‘sin6_scope_id’ field is set to or carries the interface index for the Link-Local address.
5.2. getaddrinfo()

When a node and service are successfully looked up using the getaddrinfo() call [RFC3493], a set of one or more ‘addrinfo’ structures is returned. Two of the fields within the returned ‘addrinfo’ structure(s) are the ‘ai_addr’ pointer, pointing to a ‘sockaddr’ structure, and the corresponding ‘ai_addrlen’ field, describing the size of the ‘sockaddr’ structure. These two fields are used to describe the ‘sockaddr’ structure holding one of the Internet addresses that corresponds to the node and service looked up using the getaddrinfo() call.

In the case of an IPv6 Internet address, the ‘ai_addr’ pointer will point to a ‘sockaddr_in6’ structure, which will include the ‘sin6_scope_id’ field, described previously.

Although traditional unicast DNS AAAA resource records are not prohibited from holding Link-Local addresses [RFC3596], AAAA resource records can only hold a 128 bit IPv6 address; the IPv6 scope information for the address is not stored. In this case, a AAAA resource record holding a Link-Local address is going to be ambiguous by itself; the ‘sin6_scope_id’ field value will likely be zero, indicating the "default" zone [RFC4007]. It will be necessary to supply scope information separately from getaddrinfo() to successfully use a Link-Local address stored in a unicast DNS AAAA resource record. Note that [RFC4472] recommends against storing Link-Local addresses in unicast DNS AAAA resource records.

However, multicast DNS [RFC6762] queries for hosts within the ".local" top-level domain can implicitly provide the scope for a Link-Local address received in an AAAA resource record, as multicast DNS queries for the ".local" top-level domain uses a Link-Local scope multicast address (ff02::fb). This means that both the multicast DNS client and responder must be attached to the same link. As a multicast DNS responder is required to advertise all of its valid addresses, including its Link-Local address(es), a Link-Local address received in a multicast DNS response will be present on the link over which the response was received. Consequently, a multicast DNS client module operating behind the getaddrinfo() call can supply an unambiguous Link-Local address scope value in the ‘sin6_scope_id’ field of a ‘sockaddr_in6’ structure. Other Sockets API calls that use the ‘sockaddr_in6’ structure should then work successfully with Link-Local addresses that have been supplied via multicast DNS.

Note that if getaddrinfo() returns multiple ‘addrinfo’ structures, they will be sorted as destination addresses in accordance with [RFC6724]. By default, Link-Local ‘addrinfo’ structures will appear before all other IPv6 address types, with exception to the IPv6
loopback address (::1). As applications are normally advised to step through the list in order, making successive connection attempts until one is successful, the earlier appearance of Link-Local addresses should ensure the attempted use of Link-Local addresses for application connections occurs before connections to larger scope ULA and GUA addresses are attempted. The "Happy Eyeballs" technique [RFC6555] will instead cause IPv4 to be quickly used if available, if a connection to a Link-Local address fails.

6. Security Considerations

Using Link-Local addresses, when possible, increases application security by limiting the set of possible attackers to those located on the same link as a possible victim. In many cases, these possible same link attackers will be located geographically close to a possible victim, which may further discourage attempted attacks.

7. Acknowledgements

Review and comments were provided by YOUR NAME HERE!

This memo was prepared using the xml2rfc tool.

8. Change Log [RFC Editor please remove]

draft-smith-ipv6-link-locals-apps-00, initial version, 2015-10-19

9. Informative References

[I-D.smith-6man-link-locals-off-link]
Smith, M., "Indicating Link-Local Unicast Destinations are Off-Link", draft-smith-6man-link-locals-off-link-00 (work in progress), August 2015.


Smith                    Expires April 21, 2016                 [Page 8]


Author’s Address

Mark Smith
PO BOX 521
HEIDELBERG, VIC 3084
AU

Email: markzzzsmith@gmail.com