Transport Services (taps) Working Group
Internet-Draft
Intended status: Informational
Expires: September 22, 2018

F. Gont
SI6 Networks / UTN-FRH
G. Gont
SI6 Networks
M. Garcia Corbo
SITRANS
C. Huitema
Private Octopus Inc.
March 21, 2018

Problem Statement Regarding Limitations of the Sockets API for IPv6 Address Usage
draft-gont-taps-sockets-api-limitations-00

Abstract

This identifies gaps that currently prevent systems and applications from leveraging the increased flexibility and availability of IPv6 addresses.

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 September 22, 2018.

Table of Contents

1. Introduction

IPv6 hosts typically configure a number of IPv6 addresses of different properties. For example, a host may configure one stable and one temporary address per each autoconfiguration prefix advertised on the local network. Currently, the addresses to be configured typically depend on local system policy, with the aforementioned policy being static and irrespective of the network the host attaches to. This "one size fits all" approach limits the ability of systems and applications of fully-leveraging the increased flexibility and availability of IPv6 addresses.

Each application running on a given system may have its own set of requirements or expectations for the properties of the IPv6 addresses to be employed. For example, an application meaning to offer a public service might expect to employ global stable addresses for such purpose, while a privacy-sensible client application might prefer short-lived temporary addresses, or might even expect to employ single-use ("throw-away") IPv6 addresses when connecting to public servers. However, the subtetlies associated with IPv6 addresses (and associated properties) are often ignored by application programmers and, in any case, current APIs (such as the BSD Sockets API) tend to be very limited in the amount of control

they give applications to select the most appropriate IPv6 addresses for a given task, thus limiting a programmer's ability to leverage IPv6 address availability and properties.

This document provides a problem statement by identifying and analyzing gaps that prevent systems and applications from fully-leveraging IPv6 addressing capabilities, setting the basis for further work that could fill those gaps.

2. Terminology

This document employs the definitions of "public address", "stable address", and "temporary address" from Section 2 of [RFC7721].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Default Address Selection in IPv6

Applications use system API's to select the IPv6 addresses that will be used for incoming and outgoing connections. These choices have consequences in terms of privacy, security, stability and performance.

Default Address Selection for IPv6 is specified in [RFC6724]. The selection starts with a set of potential destination addresses, such as returned by getaddrinfo(), and the set of potential source addresses currently configured for the selected interfaces. For each potential destination address, the algorithm will select the source address that provides the best route to the destination, while choosing the appropriate scope and preferring temporary addresses. The algorithm will then select the destination address, while giving a preference to reachable addresses with the smallest scope. The selection may be affected by system settings. We note that [RFC6724] only applies for outgoing connections, such as those made by clients trying to use services offered by other hosts.

We note that [RFC6724] selects IPv6 addresses from all the currently available addresses on the host, and there is currently no way for an application to indicate expected or desirable properties for the IPv6 source addresses employed for such outgoing communications. For example, a privacy-sensitive application might want that each outgoing communication instance employs a new, single-use IPv6 address, or to employ a new reusable address that is not employed or reusable by any other application on the host. Reuse of an IPv6 address by an application would allow the correlation of all network activities corresponding to such application as being performed by

the same host, while reuse of an IPv6 address by multiple different applications would allow the correlation of all such network activities as being performed by the host with such IPv6 address.

When devices provide a service, the common pattern is to just wait for connections over all addresses configured on the device. For example, applications using the BSD Sockets API will commonly bind() the listening socket to the undefined address. This long-established behavior is appropriate for devices providing public services, but may have unexpected results for devices providing semi-private services, such as various forms of peer-to-peer or local-only applications.

This behavior leads to three problems: device tracking, unexpected address discovery, and availability outside the expected scope (please see [I-D.gont-taps-address-analysis] for more details). These problems are caused in part by the limitations of available address selection API, presented in Section 5.1.

4. Current Possible Approaches for IPv6 Address Usage

4.1. Incoming communications

There are a number of ways in which a system or network may affect which address (and how) may be employed for different services and cases. Namely,

- TCP/IP stack address filtering
- Application-based address filtering
- Firewall-based address filtering

Clearly, the most elegant approach for address selection is for applications to be able to specify the properties of the addresses they are willing to employ by means of an API, such the TCP/IP stack itself can "filter" which addresses are allowed to be employed for the given service/application. This relieves the application from dealing with low level details of networking, improves portability, and avoids duplicate code in applications. However, constraints in the current APIs (see Section 5.1) may limit the ability of application progremmers for leveraging this technique.

Another possible approach is for applications to e.g. bind services to all available addresses, and perform the associated selection/ filtering at the application level. While possible this has a number of drawbacks. Firstly, it would require applications to deal with low-level networking details, require that all the associated code be

duplicated in all applications, and also negatively affect portability. Besides, performing address/selection filtering at the application level may not mitigate some possible threats. For example, port scanning will still be possible, since the aforementioned filtering will only be performed e.g. once UDP packets are received or TCP connections are established.

Finally, a firewall may be employed to filter addresses based on their intended usage. For example, a firewall may block incoming requests to all addresses except to some whitelisted addresses (such as the stable addresses of the node). This technique not only requires the use of a firewall (which may or may not be present), but also implies knowledge of the firewall regarding the desired properties of the addresses that each application/service is intended to use.

4.2. Outgoing communications

An application might be able to obtain the list of currently-configured addresses, and subsequently select an address with desired properties, and explicitly "bind" the address to the socket, to override the default source address selection.

However, this approach is problematic for a number of reasons. Firstly, there is no portable way of obtaining the list of currently-configured addresses on the local node, and even less to check for properties such "valid lifetime". Secondly, as discussed in Section 4.1, it would require application programmers to understand all the subtetiles associated with IPv6 addressing, and would also lead to duplicate code on all applications. Finally, applications would be limited to use already-configured addresses and unable to trigger the generation of new addresses where desirable (e.g. the genration of a new temporary address for this application instance or communication instance).

5. Problem Statement

This section elaborates the problem statement on IPv6 address usage. Section 5.1, Section 5.3, Section 5.2, analyze the possible root of such improper IPv6 address usage, suggesting possible future work.

5.1. Current Limitations in the Address Selection APIs

Application developers using the BSD Sockets API can "bind" a listening socket to a specific address, and ensure that the application is only reachable through that address. In theory, careful selection of the binding address could mitigate the problems described in [I-D.gont-taps-address-analysis] . Binding services to

temporary addresses could mitigate the ability of an attacker from testing for the presence of the node in the network. Binding different services to different addresses could mitigate unexpected discovery. Binding services to link local addresses or ULA could mitigate availability outside the expected scope. However, explicitly managing addresses adds significant complexity to the application development. It requires that application developers master addressing architecture subtleties, and implement logic that reacts adequately to connectivity events and address changes. Experience shows that application developers would probably prefer some much simpler solution.

In addition, we should note that many application developers use high level APIs that listen to TLS, HTTP, or some other application protocol. These high level APIs seldom provide detailed access to specific IP addresses, and typically default to listening to all available addresses.

A more advanced API could allow an application programmer to select desired properties in an address (scope, lifespan, etc.), such that the best-suitable addresses are selected, while relieving the application for low-level IPv6 addressing details. Such API might also trigger the generation of new IPv6 addresses when the specified properties would require so.

5.2. Sub-optimal IPv6 Address Configuration

Most operating systems configure the same types of addresses regardless of the current "operating mode" or "profile" of the device (e.g., device connected to enterprise network vs roaming across untrusted networks). For example, many operating systems configure both stable [RFC8064] and temporary [RFC4941] addresses on all network interfaces. However, this "one size fits all" approach tends to be sub-optimal or inappropriate for some scenarios. For example, enterprise networks typically prefer usage of only stable address, thus meaning that a network administator needs to find the means for disabling the generation of temporary addresses on all those systems that would otherwise generate them. On the other hand, some mobile devices configure both stable and temporary addresses, even when their usage pattern (client-like operation, as opposed to offering services to other nodes) would allow for the more privacy-sensible option of configuring only temporary addresses.

The lack of better tuned address configuration policies has helped the "one size fits all" approach that, as noted, may lead to suboptimal results. Advice in this area might help achieve more optional address generation policies such that IPv6 addressing capabilities are fully leveraged.

5.3. Sub-optimal IPv6 Address Usage

An application programmer, left with the question of which are the most appropriate addresses for a given usage type and application, typically resorts to the Default IPv6 Address Selection for IPv6 (see Section 3) for outgoing communications, and to accepting incoming communications on all available addresses for incoming communications. As discussed throughout this document, this leads to sub-optimal results. Besides, all applications on a node share the same pool of configured addresses, and applications are also prevented from triggering the generation of new addresses (e.g. to be employed for a particular application or communcation instance).

Guidance in this area is warranted such that applications and systems fully-leverage IPv6 addressing.

6. IANA Considerations

There are no IANA registries within this document. The RFC-Editor can remove this section before publication of this document as an RFC.

7. Security Considerations

The security and privacy implications associated with the predictability and lifetime of IPv6 addresses has been analyzed in [RFC7217] [RFC7721], and [RFC7707]. This document complements and extends the aforementioned analysis by considering other IPv6 properties such as the address scope and address usage type, and the associated tradeoffs.

8. Acknowledgements

The authors would like to thank (in alphabetical order) Francis Dupont, Tatuya Jinmei, Erik Kline, Tommy Pauly, and Dave Thaler for providing valuable comments on earlier versions of this document.

Fernando Gont would like to thank Spencer Dawkins for his guidance.

9. References

9.1. Normative References

[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/info/rfc2119>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <https://www.rfc-editor.org/info/rfc7217>.

[RFC8064] Gont, F., Cooper, A., Thaler, D., and W. Liu, "Recommendation on Stable IPv6 Interface Identifiers", RFC 8064, DOI 10.17487/RFC8064, February 2017, <https://www.rfc-editor.org/info/rfc8064>.

9.2. Informative References

[Barnes2012] Barnes, R., Altmann, R., and D. Kerr, "Mapping the Great Void Smarter scanning for IPv6", ISMA 2012 AIMS-4 - Workshop on Active Internet Measurements, February 2012, <https://www.caida.org/workshops/isma/1202/slides/aims1202_rbarnes.pdf>.

[Hein] Hein, B., "The Rising Sophistication of Network Scanning", January 2016, <http://netpatterns.blogspot.be/2016/01/the-rising-sophistication-of-network.html>.

[I-D.gont-6man-address-usage-recommendations] Gont, F., Gont, G., Corbo, M., and C. Huitema, "Problem Statement Regarding IPv6 Address Usage", draft-gont-6man-address-usage-recommendations-04 (work in progress), October 2017.

[I-D.gont-6man-non-stable-iids] Gont, F., Huitema, C., Krishnan, S., Gont, G., and M. Corbo, "Recommendation on Temporary IPv6 Interface Identifiers", draft-gont-6man-non-stable-iids-03 (work in progress), March 2018.

[I-D.gont-opsawg-firewalls-analysis] Gont, F. and F. Baker, "On Firewalls in Network Security", draft-gont-opsawg-firewalls-analysis-02 (work in progress), February 2016.

[I-D.ietf-v6ops-ula-usage-considerations] Liu, B. and S. Jiang, "Considerations For Using Unique Local Addresses", draft-ietf-v6ops-ula-usage-considerations-02 (work in progress), March 2017.

[RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <https://www.rfc-editor.org/info/rfc7707>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

Authors' Addresses

Fernando Gont
SI6 Networks / UTN-FRH
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: fgont@si6networks.com
URI: http://www.si6networks.com

Guillermo Gont
SI6 Networks
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: ggont@si6networks.com
URI: https://www.si6networks.com

Madeleine Garcia Corbo
Servicios de Informacion del Transporte
Neptuno 358
Havana City 10400
Cuba

Email: madelen.garcia16@gmail.com

Christian Huitema
Private Octopus Inc.
Friday Harbor, WA 98250
U.S.A.

Email: huitema@huitema.net
URI: http://privateoctopus.com