Network Working Group
Internet-Draft
Expires: April 25, 2004

E. Nordmark
S. Chakrabarti
Sun Microsystems, Inc.
J. Laganier
ENS Lyon / Sun Microsystems, Inc.

October 26, 2003

IPv6 Socket API for Address Selection
draft-chakrabarti-ipv6-addrselect-api-02

Abstract

The IPv6 default address selection document describes the rules for selecting default address by the system and indicates that the applications should be able to reverse the sense of system preference of address selection for that application through possible API extensions. However, no such socket API exists in the basic or advanced IPv6 socket API documents. Hence this document specifies socket level options add new flags for the getaddrinfo() API to specify preferences for address selection that modify the default address selection algorithm. The socket APIs described in this

document will be particularly useful when Mobile IPv6 is used, for IPv6 applications which want to choose between temporary and public addresses, CGA (cryptographically generated addresses) and non-CGA addresses, and applications which do not want the default preferences with respect to address scopes.

Table of Contents

1. Introduction

This document defines socket API extensions to support the non-default choice of address selection by the applications. The IPv6 default address selection [1] document has specified the rules for system default address selection for an outbound IPv6 packet. Privacy considerations [6] have introduced "public" and "temporary" addresses. IPv6 Mobility [3] introduces "home address" and "care-of-address" definitions in the mobile systems. Some applications might want to control whether large scope or small scope IPv6 addresses are preferred.

Although it is desirable to have default algorithms for address selection, an application may want to reverse certain address selection rules for efficiency and other application specific reasons. Currently IPv6 socket API extensions provide mechanism to choose a specific source address through simple bind() operation or IPV6_PKTINFO socket option [5]. Thus in order to use bind() or IPV6_PKTINFO socket option, the application itself must make sure that the source address is appropriate for the destination address (e.g., with respect to the interface used to send packets to the destination). The application also needs to make sure about the appropriate scope of source address with respect to the destination address and so on. The mechanism presented in this document allows the application to specify attributes of the source (and destination) addresses it prefers while still having the system perform the rest of address selection. For instance, if an application prefers to use care-of-address of a mobile node as the source address and if the mobile node has two care-of-addresses (one public and one temporary), then the node would select the public care-of-address by following the default address selection rule for public and temporary address.

A socket option has been deemed useful for this purpose, as it enables an application to specify addesss selection preferences on a per-socket basis. It can also provide the flexibility of enabling and disabling address selection preferences in non-connected sockets. The socket option uses a set of flags for address preferences. Since source address selection and destination address ordering need to be partially implemented in getaddrinfo() [2] the corresponding set of flags are also defined for that routine.

Thus this document introduces several flags for address selection preferences that alter the default address selection [1] for a number of cases. It analyzes the usefulness of providing API functionality for different default address selection rules; it provides API to alter only those rules that are possibly used by certain class of applications. In addition, it also considers CGA [7][8] and non-CGA source addresses when CGA addresses are available in the system. In

the future, more destination or source flags may be added to expand the API as the needs may arise.

The approach in this document is to allow the application to specify preferences for address selection and not to be able to specify hard requirements. Thus for instance, an application can set a flag to prefer a temporary source address, but if no temporary source addresses are available at the node, a public address would be chosen instead.

Specifying a 'requirement' for address selection is not adopted at the application level due to the nature of unreliable transport protocols where the failure of connect() operation may appear late in absence of the required attribute of source address in the system. This document defines a verification function which applications may choose to use before sending data on a connected socket. By "connected" socket we mean that a connect() call is done after setting setsockopt() with the preference attributes. Note that connect() can be used in UDP datagram sockets as well. The purpose of checking the validation of address after connect() call ensures the availability of the desired address type; an application using only sendto() or sendmsg() cannot guarantee the validated address at the time of sending data . The configuration of node may change or the address may expire between setsockopt() setting and sendto() or sendmsg() call.

Furthermore, the approach is to define two flags for each purpose, so that an application can specify either that it prefers 'X' or prefers 'not X', or it can choose not to set either of the flags relating to 'X' and leave it up to the system default (see section 3.1). For example, if setsockopt() with a preference to care-of-address is set, but no flag is set to indicate a choice of temporary or public address, then temporary vs. public source address selection will be determined from the default source address selection [1] rules. Thus not specifying either of "X" and "not X" leaves the "X" property of the address selection at the system default.

This document only specifies the extensions for the socket API since the socket API is already specified in RFCs [2]. The intent is that this document serve as a model for the type of address selection preferences that need to be expressable in other networking API such as those found in middleware systems and the Java environment.

2. Design Alternatives

Suggestions have been made that have flags per application instead of per socket would be more flexible. However, this design stays with per socket flags for the following reasons:

- while some system have per environment/application flags (such as environment variables in Unix systems) this might not be available in all systems which implement the socket API

- when an application links with some standard library that library, unknown to the application, might be using the socket API. Mechanisms that would provide per application flags would affect not only the application itself but also the libraries' use of the socket API with a large risk for unintended consequences.

3. Example Usages

The examples discussed here are limited to applications supporting Mobile IPv6, IPv6 Privacy Extensions and Cryptographically Generated Addresses. Address selection document [1] recommends that home addresses should be preferred over care-of-address when both are configured. However, a mobile node may want to prefer care-of-address as source address for DNS query in the foreign network as it normally means a shorter and local return path compared to the route via the mobile node's home-agent when the query contains home-address as source address. Another example is IKE application which requires care-of-address as its source address for the initial security association pair with Home Agent [3] while the mobile node boots up at the foreign network and wants to do the key exchange before a successful home-registration. Also a Mobile IPv6 aware application may want to toggle between home-address and care-of-address depending on its location and state of the application. It may also want to open different sockets and use home-address as source address for one socket and care-of-address for the others.

In a non-mobile environment, similarly an application may prefer to use temporary address as source address for certain cases. By default, the source address selection rule selects "public" address when both are available. For example, an application supporting web browser and mail-server may want to use "temporary" address for the former and "public" address for the mail-server as a mail-server may require reverse path for DNS records for anti-spam rules.

Similarly, a node may be configured to use the cryptographically generated addresses by default, as in Secure Neighbor Discovery, but an application may prefer not to use it. For instance, fping, a debugging tool which tests basic reachability of multiple destinations by sending packets in parallel, may find that the cost and time incurred in proof-of-ownership by CGA verification is not justified. On the other hand, when a node is not configured for CGA as default, an application may prefer using CGA by setting the socket option. It may subsequently verify that it is truly bound to a CGA by first calling getsockname() and then recomputing the CGA using the public key of the node.

Besides the above examples, the defined address preference flags can be used to specify or alter the system default values for largest scope of addresses as well. An application may want to use only link-local source address to contact a node with global destination address on the same link, it can do so by setting the appropriate source address preference flag in the application. By default the system would have chosen global source address. This example assumes that only link-local and global addresses are available on the nodes.

4. Changes to the Socket Interface

IPv6 Basic API [2] defines socket options for IPv6. This document adds a new socket option at the IPPROTO_IPV6 level. This socket option is called IPV6_ADDR_PREFERENCES. It can be used with setsockopt() and getsockopt() calls. This socket option takes a 32bit unsigned integer argument. The argument consists of a number of flags where each flag indicates an address selection preference which modifies one of the rules in the default address selection specification.

The following flags are defined to alter or set the default rule of source and destination address selection rules discussed in default address selection specification [1].

<netinet/in.h>.

```
IPV6_PREFER_SRC_HOME    /* Prefer Home Address as source */

IPV6_PREFER_SRC_COA     /* Prefer Care-Of_address as source */

IPV6_PREFER_SRC_TMP     /* Prefer Temporary address as source */

IPV6_PREFER_SRC_PUBLIC  /* Prefer Public address as source */

IPV6_PREFER_SRC_CGA     /* Prefer CGA address as source */

IPV6_PREFER_SRC_NONCGA  /* Prefer a non-CGA address as source */

IPV6_PREFER_SRC_LARGESCOPE /* Prefer larger scope source */

IPV6_PREFER_SRC_SMALLSCOPE  /* Prefer small(link-local) scope */
```

NOTE: No source preference flag for longest matching prefix is defined here because it is believed to be handled by the policy table defined in the default address selection specification.

Flags for altering Scope of destination addresses:

Flags corresponding to other destination address rules are not defined in this document at this point. See section 8 for more analysis and mapping of rules and different flags.

```
IPV6_PREFER_DST_LARGESCOPE /* Prefer larger scope for destination */

IPV6_PREFER_DST_SMALLSCOPE /* Prefer small scope for destination */
```

The following example illustrates how it is used on a AF_INET6 socket:

```
uint32_t flags = IPV6_PREFER_SRC_COA;

if (setsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,

    (char *) &flags, sizeof (flags)) == -1) {

             perror("setsockopt IPV6_ADDR_REFERENCES");

}
```

When the IPV6_ADDR_PREFERENCES is successfully set with setsockopt(), the option value given is used to specify address preference for any connection initiation through the socket and all subsequent packets sent via that socket. If no option is set, the system selects a default value as per default address selection algorithm or by some other equivalent means.

Setting conflicting flags at the same time results in the error EINVAL. For example, setting 'X' and 'not X' is not allowed at the same time. If flag is set as combination of 'X' and 'Y', and if 'Y' is not applicable or available in the system, then the selected address contains property of 'X' and system default for the property of 'Y'. For example, a possible valid combination of flags can be:

```
IPV6_PREFER_SRC_PUBLIC | IPV6_PREFER_SRC_LARGESCOPE
```

5. Changes to the protocol-independent nodename translation

Section 8 of Default Address Selection [1] document indicates possible implementation strategies for getaddrinfo() [2]. One of them suggests that getaddrinfo() collects available source/ destination pair from the network layer after being sorted at the network layer with full knowledge of source address selection. Another strategy is to call down to network layer to retrieve source address information and then sort the list in the context of getaddrinfo().

Thus if an application sets setsockopt() IPV6_ADDR_PREFERENCES option to alter the default address selection rules , it is required that the application calls getaddrinfo() with the corresponding AI_PREFER_* flags specified in this section. This ensures that getaddrinfo() function implementation has considered the address preference desired by the application, as the destination address selection rule is influenced by the order of source address selection. This document also defines AI flags for destination SCOPE which has direct impact on getaddrinfo() and destination address selection interaction.

There is no corresponding destination address selection rule for source address selection rule 7, in default address selection document. However, this API provides a way for an application to make sure that the source address preference set in setsockopt() is taken into account by the getaddrinfo() function. Let's consider an example to understand this scenario. DA and DB are two global destination addresses and the node has two global addresses SA and SB through interface A and B respectively. SA is a temporary address while SB is a public address. The application has set IPV6_PREFER_SRC_TMP in the setsockopt() flag. The route to DA points to interface A and route to DB points to interface B. Thus when AI_PREFER_SRC_TMP is set , getaddrinfo() returns DA before DB and SA before SB likewise. Similarly, getaddrinfo() returns DB before DA when AI_PREFER_SRC_PUBLIC is set in this example. Thus the source address preference is taking effect into destination address selection and as well as source address selection by the getaddrinfo() function.

The following numerical example clarifies the above further.

Imagine a host with two addresses:

```
1234::1:1 public

9876::1:2 temporary
```

The destination has the following two addresses:

1234::9:3

9876::9:4

By default getaddrinfo() will return the destination addresses in the order

1234::9:3

9876::9:4

because the public source is preferred and 1234 matches more bits with the public source address. On the other hand, if AI_PREFER_SRC_TMP is set, getaddrinfo will return the addresses in the reverse order since the temporary source address will be preferred.

The following flags are added for the ai_flags in addrinfo data structure defined in Basic IPv6 Socket API Extension [2].

```
AI_PREFER_SRC_HOME        /* Prefer Home Address */

AI_PREFER_SRC_COA         /* Prefer COA  */

AI_PREFER_SRC_TMP         /* Prefer Temporary Address */

AI_PREFER_SRC_PUBLIC      /* Prefer Public Address */

AI_PREFER_SRC_CGA         /* Prefer CGA Address */

AI_PREFER_SRC_NONCGA      /* Prefer address other than CGA */

AI_PREFER_SRC_LARGESCOPE  /* Prefer large(global) scope */

AI_PREFER_SRC_SMALLSCOPE  /* Prefer small(local) scope */

AI_PREFER_DST_LARGESCOPE  /* Prefer large(global) scope dest.  */

AI_PREFER_DST_SMALLSCOPE  /* Prefer small(local) scope dest.*/
```

The above flags are ignored for the AF_INET address family as the address selection algorithm defined in section 5 of [1] only applies to the IPv6 addresses.

If conflicting flags such as AI_PREFER_SRC_HOME and AI_PREFER_SRC_COA are set, the getaddrinfo() fails with an error EAI_BADFLAGS [2].

Some valid sequences of flags are:

```
AI_PREFER_SRC_HOME | AI_PREFER_SRC_PUBLIC

AI_PREFER_SRC_COA  | AI_PREFER_SRC_PUBLIC

AI_PREFER_SRC_HOME | AI_PREFER_SRC_CGA

AI_PREFER_SRC_HOME | AI_PREFER_SRC_NONCGA

AI_PREFER_SRC_COA  | AI_PREFER_SRC_CGA

AI_PREFER_SRC_COA  | AI_PREFER_SRC_NONCGA

AI_PREFER_SRC_LARGESCOPE | AI_PREFER_DST_LARGESCOPE

AI_PREFER_SRC_SMALLSCOPE | AI_PREFER_DST_SMALLSCOPE

AI_PREFER_SRC_LARGESCOPE | AI_PREFER_DST_LARGESCOPE |
AI_PREFER_SRC_PUBLIC
```

All the constants mentioned in this section for ai_flags are defined in <netdb.h>.

6. Application Requirements

An application SHOULD call getsockopt() prior calling setsockopt() to a particular address preference, in order to save the existing system default values or the current values of the address preference flags. However, setsockopt() with a flag value 0 resets the address selection to the system default policy.

Example:

```
uint32_t save_flag, flags;

int optlen = sizeof (save_flag);

/* Save the existing IPv6_ADDR_PREFERENCE FLAG now */

if (getsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,

      &save_flag, &optlen) == -1 {

             perror("getsockopt IPV6_ADDR_REFERENCES");

}

flags = save_flag;

flags &= ~IPV6_PREFER_SRC_PUBLIC;

flags |= IPV6_PREFER_SRC_TMP;

if (setsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,

    (char *) &flags, sizeof (flags)) == -1) {

             perror("setsockopt IPV6_ADDR_REFERENCES");

}
```

Application MUST not set conflicting flags; the only conflicts that are checked for are flag X and flag not-X being set at the same time. Example of conflicting flags :
IPV6_PREFER_SRC_TMP | IPV6_PREFER_SRC_PUBLIC.

In order to allow different implementations to do different parts of address selection in getaddrinfo() and in the protocol stack, this specification requires that applications set the same flags when calling getaddrinfo() and when calling setsockopt(). For example, if the application sets IPV6_PREFER_SRC_COA flag, it must use

AI_PREFER_SRC_COA flag when calling getaddrinfo(). If applications are not setting the same flags the behavior of the implementation is undefined.

It is envisioned that Mobile IPv6 applications may want to choose Care-of-Address as source for short transaction (for efficiency) while roaming, but still keep Home address as source address for long lived communication for address stability. Thus it is recommended that applications take this idea into consideration and use the source address selection API for home-address and care-of -address selection appropriately. Similarly, an application may choose to set IPV6_PREFER_SRC_COA flag for datagram services; it uses home-address as source when at home and uses care-of- address outside home-network for short datagram transactions. This is an advantage of having flexibility of "preference" vs. "requirement".

7. Implementation Notes

- If either bind() or IPV6_PKTINFO socket option is set with a specific source address in the same application along with the address preferenc e socket option, then bind() or IPV6_PKTINFO option takes precedence.

- setsockopt() and getsockopt() SHOULD ignore any address preference flags for type of addresses that are not supported in the system. The socket option calls should return error when invalid flag values are passed to them. The invalid flag values are: flag X and flag not-X (set at the same time), some flag that is not known to the implementation.

- If an implementation supports both streams and datagram sockets, it should implement the address preference mechanism API described in this document on both cases.

- Implementation supporting this API must implement both AI flags and socket option flags processing for portability of applications.

- An implementation may choose to set the following flags by default on the system. However, an implementation may choose to clear the address preference flags by default indicating that system is following default address selection rules.Thus it sets the address preference flags only when it is set by the application. Thus the change in address preference flag is only visible by that application.

  IPV6_PREFER_SRC_HOME

  IPV6_PREFER_SRC_PUBLIC

  IPV6_PREFER_SRC_SMALLSCOPE

  IPV6_PREFER_SRC_CGA

  IPV6_PREFER_DST_SMALLSCOPE

8. Mapping to Default Address Selection Rules

This API defines only those flags that are deemed to be useful by the applications to alter default address selection rules. Thus we discuss the mapping of each set of flags to the corresponding rule number in the address selection document[1].

Source address selection rule #4 (prefer home address):

IPV6_PREFER_SRC_HOME

IPV6_PREFER_SRC_COA

AI_PREFER_SRC_HOME

AI_PREFER_SRC_COA

Source address selection rule #7 (prefer public address) :

IPV6_PREFER_SRC_PUBLIC

IPV6_PREFER_SRC_TMP

AI_PREFER_SRC_PUBLIC

AI_PREFER_SRC_TMP

Source address selection rule #2 (prefer appropriate scope):

IPV6_PREFER_SRC_LARGESCOPE

IPV6_PREFER_SRC_SMALLSCOPE

AI_PREFER_SRC_LARGESCOPE

AI_PREFER_SRC_SMALLSCOPE

Destination address selection rule #8 (prefer smaller scope):

IPV6_PREFER_DST_LARGESCOPE

IPV6_PREFER_DST_SMALLSCOPE

AI_PREFER_DST_LARGESCOPE

AI_PREFER_DST_SMALLSCOPE

Other destination rules (#4-prefer home address; #7-prefer native

interfaces) could have been applicable. But the problem is that the local system does not know about whether a destination address is a tunnel-address for destination rule #7. It can only know for sure if the destination address is one of its own. The flags defined for source address selection rule #4 (prefer home address) should also take care of destination address selection rule #4. Thus at this point, it was decided not to define flags for these destination rules.

Other source address rules (that are not mentioned here) were also deemed not applicable for changing its default notion per-application basis.

9. IPv4-mapped IPv6 Addresses

IPv4-mapped IPv6 addresses are supported in this API. In some cases the IPv4-mapped addresses may not make much sense because the attributes are IPv6 specific. For example, IPv6 temporary addresses are not the same as private IPv4 addresses. However, the IPv4 mapped-address support may be useful for mobile home address and care-of-address. At this point it is not understood whether this API has any value to IPv4 addresses or AF_INET family of sockets.

10. Validation function for source address

Sometimes an application may have a requirement to set a specific source address without which it chooses to fail. In that situation, 'preferred' addresses do not guarantee the application requirement. An application which requires to set a specific type of source address must verify that the system indeed has a valid source address for the desired source address type. A validation function is defined for this purpose:

```
<netinet/in.h>.

boolean_t  inet6_is_srcaddr(struct sockaddr_in6 * srcaddr,
               uint32_t flags)
```

Where the flags contain the specified source preference flags. The function expects a non-NULL input for srcaddr. It returns true when srcaddr corresponds to a valid address in the node and that address type satisfies the preference flag. If srcaddr input value does not correspond to any address in the node or it does not match an address which satisfy the preferences indicated, the function returns false. Currently, the validation function seems meaningful only for IPV6_PREFER_SRC_TMP, IPV6_PREFER_SRC_PUBLIC and IPV6_PREFER_SRC_[NON]CGA flags. See the section on "Application Requirements" for usage of preference flags in Mobile IPv6 applications. The scope preference flags do not guarantee validation of largest scope when more than two scopes are configured. Thus the above temporary/public and CGA/non-CGA flags are the predictable choices for the validation function.

sockaddr_in6 structure must contain AF_INET6 as sin6_family. It also must contain the scope_id information if the source address is a link-local address.

This function should be able to handle multiple valid flags combination as its second parameter. Invalid flag values result in false return value.

The verification function can be useful for both TCP and UDP socket applications that use connect().

11. Security Considerations

This document conforms to the same security implications as specified in IPv6 Basic Socket API [2] document. Allowing applications to specify a preference for temporary addresses provides per-application (and per-socket) ability to use the privacy benefits of the temporary addresses.

12. Changes from previous version of draft

- Changed IPV6_SRC_PREFERENCES option to IPV6_ADDR_PREFERENCES to include destination address preference for scope and for further future enhancement which may include both source and destination addresses.

- Added implementation and application requirements.

- Removed IPV6_PREFER_SRC_NATIVE and IPV6_PREFER_SRC_TUNNEL flags as there is no corresponding source address rule in RFC3484. Moreover it doesn't seem to make sense to add preference flags for this destination addresses since:

  - the local system doesn't in general know whether there is a tunnel at the destination end and

  - in the case (6to4) where the local system can tell there will be a tunnel for a destination address the default policy table already has a rule (for the 6to4 prefix).

  Perhaps there should have been a source rule for tunnel vs. native interface in default address selection specification in which case it might have made sense to add a preference flag for that.

- Added section on default address selection rule mapping.

- Added comments on using JAVA API.

- Added four new flags for destination scoped addresses as some working group members felt the requirement of altering default destination address scope.

13. Acknowledgments

The authors like to thank members of mobile-ip and ipv6 working groups for useful discussion on this topic. Richard Draves and Dave Thaler suggested that getaddrinfo also needs to be considered along with the new socket option. Gabriel Montenegro suggested that CGAs may also be considered in this document. Thanks to Alain Durand, Renee Danson, Alper Yegin, Francis Dupont, Michael Hunter, Sebastien Roy, Robert Elz, Jinmei Tatuya, Pekka Savola, Itojun, Jim Bound, Jeff Boote and Mika Liljeberg for useful discussions and suggestions.

Normative References

[1] Draves, R., "Default Address Selection for IPv6", RFC 3484, August 2002.

[2] Gilligan, R., Thomson, S., Bound, J., McCann, J. and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, March 2003.

Informative References

[3] Johnson, D., Perkins, C. and J. Arkko, "Mobility Support in IPv6", draft-ietf-mobileip-ipv6-24.txt (work in progress), June 2003.

[4] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6), Specification", RFC 2460, December 1998.

[5] Stevens, W., Thomas, M., Nordmark, E. and T. Jinmei, "Advanced Sockets API for IPv6", RFC 3542, May 2003.

[6] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[7] Aura, T., "Cryptographically Generated Addresses (CGA)", draft-ietf-send-cga-01.txt (work in progress), August 2003.

[8] Montenegro, G. and C. Castelluccia, "Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses.", NDSS 2002, February 2002.

Authors' Addresses

Erik Nordmark
Sun Microsystems, Inc.
180, avenue de l'Europe
38334 Saint Ismier CEDEX
France

EMail: Erik.Nordmark@Sun.COM

Samita Chakrabarti
Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
USA

EMail: Samita.Chakrabarti@Sun.COM

Julien Laganier
ENS Lyon / Sun Microsystems, Inc.
180, avenue de l'Europe
38334 Saint Ismier CEDEX
France

EMail: Julien.Laganier@Sun.COM

Appendix A. Intellectual Property Statement

This document only defines a source preference flag to choose Cryptographically Generated Address (CGA) as source address when applicable. CGA are obtained using public keys and hashes to prove address ownership. Several IPR claims have been made about such methods.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.