

Network Working Group  
Internet-Draft  
Expires: April 23, 2007

E. Nordmark  
Sun Microsystems, Inc.  
S. Chakrabarti  
Azair Networks  
J. Laganier  
DoCoMo Euro-Labs  
October 20, 2006

**IPv6 Socket API for Address Selection**  
**draft-chakrabarti-ipv6-addrselect-api-04**

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April 23, 2007.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The IPv6 default address selection document [[RFC3484](#)] describes the rules for selecting source and destination IPv6 addresses, and indicates that applications should be able to reverse the sense of some of the address selection rules through some unspecified API. However, no such socket API exists in the basic [[RFC3493](#)] or advanced

[RFC3542] IPv6 socket API documents. This document fills that gap by specifying new socket level options and flags for the `getaddrinfo()` API to specify preferences for address selection that modify the default address selection algorithm. The socket API described in this document will be particularly useful for IPv6 applications that want to choose between temporary and public addresses, and for Mobile IPv6 aware applications that want to use the care-of address for communication.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">2.</a>	Definition Of Terms . . . . .	<a href="#">6</a>
<a href="#">3.</a>	Design Alternatives . . . . .	<a href="#">7</a>
<a href="#">4.</a>	Example Usages . . . . .	<a href="#">8</a>
<a href="#">5.</a>	Additions to the Socket Interface . . . . .	<a href="#">9</a>
<a href="#">6.</a>	Additions to the protocol-independent nodename translation . .	<a href="#">11</a>
<a href="#">7.</a>	Application Requirements . . . . .	<a href="#">15</a>
<a href="#">8.</a>	Implementation Notes . . . . .	<a href="#">17</a>
<a href="#">9.</a>	Mapping to Default Address Selection Rules . . . . .	<a href="#">18</a>
<a href="#">10.</a>	IPv4-mapped IPv6 Addresses . . . . .	<a href="#">19</a>
<a href="#">11.</a>	Validation function for source address . . . . .	<a href="#">20</a>
<a href="#">12.</a>	IANA Considerations . . . . .	<a href="#">22</a>
<a href="#">13.</a>	Security Considerations . . . . .	<a href="#">23</a>
<a href="#">14.</a>	Acknowledgments . . . . .	<a href="#">24</a>
<a href="#">15.</a>	References . . . . .	<a href="#">25</a>
<a href="#">15.1.</a>	Normative references . . . . .	<a href="#">25</a>
<a href="#">15.2.</a>	Informative references . . . . .	<a href="#">25</a>
<a href="#">Appendix A.</a>	Per Packet Address Selection Preference . . . . .	<a href="#">26</a>
<a href="#">Appendix B.</a>	Changes from previous version of draft . . . . .	<a href="#">28</a>
<a href="#">Appendix C.</a>	Intellectual Property Statement . . . . .	<a href="#">29</a>
	Authors' Addresses . . . . .	<a href="#">30</a>
	Intellectual Property and Copyright Statements . . . . .	<a href="#">31</a>



## 1. Introduction

[RFC3484] specifies the default address selection rules for IPv6 [RFC2460]. This document defines socket API extensions that allow applications to override the default choice of address selection. Privacy considerations [RFC3041] have introduced "public" and "temporary" addresses. IPv6 Mobility [RFC3775] introduces "home address" and "care-of address" definitions in the mobile systems.

The default address selection rules in [RFC3484] in summary are that a public address is preferred over a temporary address, that a mobile IPv6 home address is preferred over a care-of address, and that a larger scope address is preferred over a smaller scope address. Although it is desirable to have default rules 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 mechanisms to choose a specific source address through simple `bind()` operation or `IPV6_PKTINFO` socket option [RFC3542]. However, 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 verify the appropriateness of the source address scope with respect to the destination address and so on. This can be quite complex for the application, since in effect it needs to implement all the default address selection rules in order to change its preferences with respect to one of the rules.

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 the address selection rules. For instance, if an application specifies that it prefers to use a care-of address over a home address as the source address and if the host has two care-of addresses, one public and one temporary, then the host would select the public care-of address by following the default address selection rule for preferring a public over a temporary address.

A socket option has been deemed useful for this purpose, as it enables an application to specify address selection preferences on a per-socket basis. It can also provide the flexibility of enabling and disabling address selection preferences in non-connected (UDP) sockets. The socket option uses a set of flags for specifying address selection preferences. Since the API should not assume a particular implementation method of the address selection [RFC3484] in the network layer or in `getaddrinfo()`, the corresponding set of



flags are also defined for `getaddrinfo()`.

As a result, this document introduces several flags for address selection preferences that alter the default address selection [[RFC3484](#)] for a number of rules. 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 classes of applications. In addition, it also considers CGA [[RFC3972](#)] 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: 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 hard requirements for address selection would be problematic for several reasons. The major one is that in the vast majority of cases the application would like to be able to communicate even if an address with the 'optimal' attributes is not available. For instance, an application that performs very short, e.g., UDP, transactional exchanges (e.g. DNS queries), might prefer to use a care-of address when running on a mobile host which is away from home since this provides a short round-trip time in many cases. But if the application is running on a mobile host that is at home, or running on a host which isn't providing Mobile IPv6, then it doesn't make sense for the application to fail due to no care-of address being available. Also, in particular when using UDP sockets and the `sendto()` or `sendmsg()` primitives, the use of hard requirements would have been problematic, since the set of available IP addresses might very well have changed from when the application called `getaddrinfo()` until it called `sendto()` or `sendmsg()`, which would introduce new failure modes.

For the few applications that have hard requirements on the attributes of the IP addresses they use, this document defines a verification function which allows such applications to properly fail to communicate when their address selection requirements are not met.

Furthermore, the approach is to define two flags for each rule that can be modified, so that an application can specify its preference for addresses selected as per the rule, the opposite preference (i.e. an address selected as per the rule reverted), or choose not to set either of the flags relating to that rule and leave it up to the system default (see [section 3.1](#)). This approach allows different



implementations to have different system defaults, and works with `getaddrinfo()` as well as `setsockopt()`. (For `setsockopt` a different approach could have been chosen, but that would still require this approach for `getaddrinfo`.)

This document specifies extensions only to the Basic IPv6 socket API specified in [[RFC3493](#)]. The intent is that this document serves as a model for expressing preferences for attributes of IP addresses, that also need to be expressible in other networking API such as those found in middleware systems and the Java environment.



## **2. Definition Of Terms**

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 [[RFC2119](#)].

Opposite flags:

Each flag expressing an address preference has a so called "opposite flag" expressing the opposite preference:

home address preference flag is the opposite of the care-of address preference flag.

temporary address preference flag is the opposite of the public address preference flag.

CGA address preference flag is the opposite of the non-CGA address preference flag.

Contradictory flags:

Any combination of flags including both a flag expressing a given address preference and a flag expressing the opposite preference constitutes Such flags are contradictory by definition of their usefulness with respect to source address selection. For example, consider a set of flags including both the home address preference flag and the care-of address preference flag. When considering source address selection, the selected address can be a home address, or a care-of address, but it cannot be both at the same time. Hence, to prefer an address which is both a home address, and a care-of address, is contradictory.



### **3. Design Alternatives**

Some suggested to have per-application flags instead of per-socket and per-packet flags. However, this design stays with per-socket and per-packet flags for the following reasons:

- o While some systems 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.
- o When an application links with some standard library, that library might use the socket API, while the application is unaware of that fact. Mechanisms that would provide per application flags may affect not only the application itself but also the libraries, hence creating risks of unintended consequences.

Instead of the pair of 'flag' and 'opposite flag' for each rule that can be modified, the socket option could have been defined to use a single 'flag' value for each rule. This would still have allowed different implementations to have different default settings as long as the applications were coded to first retrieve the default setting (using `getsockopt()`), and then clear or set the 'flag' according to their preferences, and finally set the new value with `setsockopt()`.

But such an approach would not be possible for `getaddrinfo()` because all the preferences would need to be expressible in the parameters that are passed with a single `getaddrinfo()` call. Hence, for consistency, the 'flag' and 'opposite flag' approach is used for both `getaddrinfo()` and `setsockopt()`.



#### **4. Example Usages**

The examples discussed here are limited to applications supporting Mobile IPv6, IPv6 Privacy Extensions and Cryptographically Generated Addresses. Address selection document [[RFC3484](#)] 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 [[RFC3775](#)] 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 Cryptographically Generated Addresses [[RFC3972](#)] by default, as in Secure Neighbor Discovery [[RFC3971](#)], but an application may prefer not to use it. For instance, fping [[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 corresponding preference.



## 5. Additions to the Socket Interface

The IPv6 Basic Socket API [[RFC3493](#)] defines socket options for IPv6. To allow applications to influence address selection mechanisms 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 to set and get the address selection preferences affecting all packets sent via a given socket. The socket option value (optval) is a 32-bits 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 [[RFC3484](#)].

```
#include <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 */
```

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.

The following example illustrates usage on an AF\_INET6 socket:

```
uint32_t flags = IPV6_PREFER_SRC_COA;

if (setsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,
    (const void *) &flags, sizeof (flags)) == -1) {
    perror("setsockopt IPV6_ADDR_PREFERENCES");
}
```

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 contradictory flags at the same time results in the error EINVAL. For example, contradictory combinations of flags are:





IPV6\_PREFER\_SRC\_PUBLIC | IPV6\_PREFER\_SRC\_TMP  
IPV6\_PREFER\_SRC\_HOME | IPV6\_PREFER\_SRC\_COA  
IPV6\_PREFER\_SRC\_HOME | IPV6\_PREFER\_SRC\_COA | IPV6\_PREFER\_SRC\_TMP  
IPV6\_PREFER\_SRC\_CGA | IPV6\_PREFER\_SRC\_NONCGA  
etc.

If a flag set includes a combination of 'X' and 'Y', and if 'Y' is not applicable or available in the system, then the selected address has attribute 'X' and system default for the attribute 'Y'. For example, a possible valid combination of flags can be:

IPV6\_PREFER\_SRC\_PUBLIC | IPV6\_PREFER\_SRC\_HOME



## **6. Additions to the protocol-independent nodename translation**

[Section 8](#) of Default Address Selection [[RFC3484](#)] document indicates possible implementation strategies for `getaddrinfo()` [[RFC3493](#)]. 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 the network layer to retrieve source address information and then sort the list in the context of `getaddrinfo()`.

This implies that `getaddrinfo()` should be aware of the address selection preferences of the application, since `getaddrinfo()` is independent of any socket the application might be using.

Thus if an application alters the default address selection rules by using `setsockopt()` with the `IPV6_ADDR_PREFERENCES` option, the application should also use the corresponding flags with its `getaddrinfo()` call. A new `ai_flag` is introduced (`AI_PREFERENCE`) to indicate the system to look for source address selection hints specified in the `ai_eflags`. If `AI_PREFERENCES` is set, `ai_eflags` MUST be set to the `IPV6_PREFER_SRC_*` values defined in this document. Note that the additional field for extended flags are added at the bottom of the `addrinfo` structure in order to preserve binary compatibility of the new functionality with the old applications which use the existing `addrinfo` data structure. Addition of extended flags field provides the designers freedom of adding more flags as necessary without crowding the valuable bit space in the `ai_flags` field.

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 source 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_PREFERENCES` in `ai_flags` and `IPV6_PREFER_SRC_TMP` in `ai_eflags` are set, `getaddrinfo()` returns DA before DB in the list of destination addresses and thus SA will be used to communicate with the destination DA. Similarly, `getaddrinfo()` returns DB before DA when `AI_PREFERENCES` and `ai_eflags` are set to `IPV6_PREFER_SRC_PUBLIC`. Thus the source address preference is taking effect into destination address selection and as well as source address selection by the



getaddrinfo() function.

```
struct addrinfo {
    int ai_flags;           /* input flags */
    int ai_family;         /* protocol family for socket */
    int ai_socktype;       /* socket type */
    int ai_protocol;       /* protocol for socket */
    socklen_t ai_addrlen;   /* length of socket address */
    char *ai_canonname;     /* canonical name for hostname */
    struct sockaddr *ai_addr; /* socket address for socket */
    struct addrinfo *ai_next; /* pointer to next in list */
    int ai_eflags;          /* Extended flags for special usage */
};
```

Example of usage:

```
struct addrinfo hints, *ai;
uint32_t preferences;

preferences = IPV6_PREFER_SRC_TMP;

hints.ai_flags |= AI_PREFERENCES;
hints.ai_eflags = preferences; /* Chosen address preference flag */
/* Fill in other hints fields */

getaddrinfo(...,&hints,. &ai..);

/* Loop over all returned addresses and do connect */
while (ai != NULL) {
    s = socket(ai->ai_family, ...);
    setsockopt(s, IPV6_ADDR_PREFERENCES, (void *) &preferences,
               sizeof (preferences));
    if (connect(s, ai->ai_addr, ai->ai_addrlen) == -1){
        close (s);
        ai = ai->ai_next;
        continue;
    }
    break;
}

freeaddrinfo(ai);
```

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_flags` is set to `AI_PREFERENCES` and `ai_eflags` to `IPV6_PREFER_SRC_TMP`, `getaddrinfo` will return the addresses in the reverse order since the temporary source address will be preferred.

The following new flag is added for the `ai_flags` in `addrinfo` data structure defined in Basic IPv6 Socket API Extension [[RFC3493](#)].

`AI_PREFERENCES /* address preference flag */`

If `AI_PREFERENCE` flag is set in `ai_flags`, then the `getaddrinfo()` implementation MUST look for `ai_eflags` values. `AI_PREFERENCE` flag is only meaningful if the extended flag sets (`ai_eflags`) are properly set. This document defines IPv6 source address preference values (`IPV6_PREFER_SRC_*`), so that `getaddrinfo()` can return matching destination addresses corresponding to the source address preferences. By default, `AI_PREFERENCES` are not set. If `AI_PREFERENCES` are set in `ai_flags` and `ai_eflags` are 0 or undefined, then `AI_PREFERENCES` is ignored.

Currently `AI_PREFERENCES` is defined for `AF_INET6` sockets only. But its usage should be extendable to other socket families as appropriate.

If contradictory flags such as `IPV6_PREFER_SRC_HOME` and `IPV6_PREFER_SRC_COA` are set in `ai_eflags`, the `getaddrinfo()` fails with an error `EAI_BADFLAGS` [[RFC3493](#)].

Valid combination of address selection flags are defined below:





IPV6\_PREFER\_SRC\_HOME | IPV6\_PREFER\_SRC\_PUBLIC

IPV6\_PREFER\_SRC\_COA | IPV6\_PREFER\_SRC\_PUBLIC

IPV6\_PREFER\_SRC\_HOME | IPV6\_PREFER\_SRC\_CGA

IPV6\_PREFER\_SRC\_HOME | IPV6\_PREFER\_SRC\_NONCGA

IPV6\_PREFER\_SRC\_COA | IPV6\_PREFER\_SRC\_CGA

IPV6\_PREFER\_SRC\_COA | IPV6\_PREFER\_SRC\_NONCGA

## 7. Application Requirements

An application should call `getsockopt()` prior calling `setsockopt()` if the application needs to be able to restore the socket back to the system default preferences. Note, this is suggested for portability. An application which does not have this requirement can just use `getaddrinfo()` while specifying its preferences, followed by:

```
uint32_t flags = IPV6_PREFER_SRC_TMP;

if (setsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,
              (void *) &flags, sizeof (flags)) == -1) {
    perror("setsockopt IPV6_ADDR_PREFERENCES");
}
```

An application which needs to be able to restore the default settings on the socket would instead do this:

```
uint32_t save_flags, flags;
int optlen = sizeof (save_flags);

/* Save the existing IPV6_ADDR_PREFERENCE flags now */

if (getsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,
              (void *) &save_flags, &optlen) == -1 {
    perror("getsockopt IPV6_ADDR_PREFERENCES");
}

/* Set the new flags */
flags = IPV6_PREFER_SRC_TMP;
if (setsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,
              (void *) &flags, sizeof (flags)) == -1) {
    perror("setsockopt IPV6_ADDR_PREFERENCES");
}

/*
 *
 * Do some work with the socket here.
 *
 */

/* Restore the flags */

if (setsockopt(s, IPPROTO_IPV6, IPV6_ADDR_PREFERENCES,
              (void *) &save_flags, sizeof (save_flags)) == -1) {
    perror("setsockopt IPV6_ADDR_PREFERENCES");
}
```



Applications MUST not set contradictory flags at the same time.

Example of contradictory 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 semantically equivalent flags when calling `getaddrinfo()` and when calling `setsockopt()`. For example, if the application sets the `IPV6_PREFER_SRC_COA` flag, it MUST use the same for `ai_eflag` when calling `getaddrinfo()`. If applications are not setting the semantically equivalent flags the behavior of the implementation is undefined.



## 8. Implementation Notes

- o Within the same application, if a specific source address is set by either `bind()` or `IPV6_PKTINFO` socket option, while at the same time an address selection preference is expressed with the `IPV6_ADDR_PREFERENCES` socket option, then the source address setting carried by `bind()` or `IPV6_PKTINFO` takes precedence over the address selection setting.
- o `setsockopt()` and `getaddrinfo()` should silently ignore any address preference flags that are not supported in the system. For example, a host which does not implement Mobile IPv6, should not fail `setsockopt()` or `getaddrinfo()` that specify preferences for home or care-of addresses. The socket option calls should return error (-1) and set `errno` to `EINVAL` when contradictory flags values are passed to them.
- o If an implementation supports both stream and datagram sockets, it should implement the address preference mechanism API described in this document on both types of sockets.
- o An implementation supporting this API MUST implement both `getaddrinfo()` extension flags and socket option flags processing for portability of applications.
- o The following flags are set as default values on a system (which is consistent with [RFC3484](#) defaults):

`IPV6_PREFER_SRC_HOME`

`IPV6_PREFER_SRC_PUBLIC`

`IPV6_PREFER_SRC_CGA`



## **9. 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 [[RFC3484](#)].

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

IPV6\_PREFER\_SRC\_HOME (default)

IPV6\_PREFER\_SRC\_COA

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

IPV6\_PREFER\_SRC\_PUBLIC (default)

IPV6\_PREFER\_SRC\_TMP

At this time, this document does not define flags to alter source address selection rule #2 (prefer appropriate scope for destination) and destination address selection rule #8 (prefer smaller scope) as the implementers felt that there were no practical applications that can take advantage of reverting the scoping rules of IPv6 default address selection. Flags altering other destination address selection rules (#4, prefer home address and #7, prefer native transport) could have applications, but the problem is that the local system cannot systematically determine whether a destination address is a tunnel address for destination rule #7 (although it can when the destination address is one of its own, or can be syntactically recognized as a tunnel address, e.g., a 6to4 address.) 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 on a per-application basis.





## **10. IPv4-mapped IPv6 Addresses**

IPv4-mapped IPv6 addresses for AF\_INET6 sockets are supported in this API. In some cases the application of IPv4-mapped addresses are limited because the API attributes are IPv6 specific. For example, IPv6 temporary addresses and cryptographically generated addresses have no IPv4 counterparts. Thus the IPV6\_PREFER\_SRC\_TMP or IPV6\_PREFER\_SRC\_CGA are not directly applicable to an IPv4-mapped IPv6 address. However, the IPv4-mapped address support may be useful for mobile-IPv4 applications shifting source address between the home address and the care-of address. Thus the IPV6\_PREFER\_SRC\_COA and IPV6\_PREFER\_SRC\_HOME are applicable to an IPv4-mapped IPv6 address. At this point it is not understood whether this API has any value to IPv4 addresses or AF\_INET family of sockets.



**11. Validation function for source address**

Sometimes an application may have a requirement to only use addresses with some particular attribute, and if no such address is available the application should fail to communicate instead of communicating using the 'wrong' address. In that situation, address selection preferences do not guarantee that the application requirements are met. Instead, the application has to explicitly verify that the chosen address satisfies its requirements using a validation function. Such an application would go through the following steps:

1. The application specifies one or more IPV6\_PREFER\_SRC\_\* flags and AI\_PREFERENCE ai\_flags with getaddrinfo().
2. The application specifies the same IPV6\_PREFER\_SRC\_\* flags with setsockopt()
3. The application calls connect(). This applies even for datagram (UDP) sockets, as the connect call results in the stack selecting a source address, for TCP as well as UDP.
4. Retrieve the selected source address using the getsockname() API call.
5. Verify with the validation function that the retrieved address is satisfactory as specified below. If not, abort the communication e.g., by closing the socket.

The verification of temporary vs. public, home vs. care-of, CGA vs. not, are performed by a new validation function defined for this purpose:

```
#include <netinet/in.h>

short inet6_is_srcaddr(struct sockaddr_in6 *srcaddr,
                      uint32_t flags);
```

Where the flags contains the specified IPV6\_PREFER\_SRC\_\* source preference flags, and the srcaddr is a non-NULL pointer to a sockaddr\_in6 structure initialized as follows:

- o sin6\_addr is a 128-bit IPv6 address of the local node.
- o sin6\_family MUST be set to AF\_INET6.
- o sin6\_scope\_id MUST be set if the address is link-local.

inet6\_is\_srcaddr() is defined to return three possible values (0, 1,



-1): The function returns true (1) when the IPv6 address corresponds to a valid address in the node and satisfies the given preference flags. If the IPv6 address input value does not correspond to any address in the node or if the flags are not one of the valid preference flags, it returns a failure (-1). If the input address does not match an address which satisfies the preference flags indicated, the function returns false (0.)

This function can handle multiple valid preference flags combination as its second parameter, for example `IPV6_PREFER_SRC_COA | IPV6_PREFER_SRC_TMP`, which means that all flags MUST be satisfied for the result to be true. Contradictory flag values result in a false return value.

The function will return true for `IPV6_PREFER_SRC_HOME` even if the host is not implementing mobile IPv6, as well as for a mobile node which is at home (i.e., does not have any care-of address).



## **12. IANA Considerations**

This document has no IANA considerations.



### **13. Security Considerations**

This document conforms to the same security implications as specified in the Basic IPv6 socket API [[RFC3493](#)]. 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. The setting of certain address preferences (e.g. not using a CGA address, or not using a temporary address) may be restricted to privileged processes because of security implications.

#### **14. 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, Keiichi Shima, Michael Hunter, Sebastien Roy, Robert Elz, Pekka Savola, Itojun, Jim Bound, Steve Cipoli, Jeff Boote, Steve Cipolli, Vlad Yasevich and Mika Liljeberg for useful discussions and suggestions. Thanks to Brian Haberman, Brian Haley, Bob Gilligan, Jack McCann, Jim Bound and Jinmei Tatuya for the review of the document and the suggestions for improvement.



## **15. References**

### **15.1. Normative references**

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", [RFC 3484](#), February 2003.
- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", [RFC 3493](#), February 2003.

### **15.2. Informative references**

- [FPING] "Fping - a program to ping hosts in parallel", Online web site <http://www.fping.com>.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", [RFC 2460](#), December 1998.
- [RFC3041] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", [RFC 3041](#), January 2001.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", [RFC 3542](#), May 2003.
- [RFC3775] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", [RFC 3775](#), June 2004.
- [RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", [RFC 3971](#), March 2005.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", [RFC 3972](#), March 2005.



## **Appendix A. Per Packet Address Selection Preference**

This document discusses setting source address selection preferences on a per socket basis with the new `IPV6_ADDR_PREFERENCES` socket option used in `setsockopt()`. The document does not encourage setting the source address selection preference on a per packet basis through the use of ancillary data objects with `sendmsg()`, or `setsockopt()` with unconnected datagram sockets.

Per-packet source address selection is expensive as the system will have to determine the source address indicated by the application preference before sending each packet, while `setsockopt()` address preference on a connected socket makes the selection once and uses that source address for all packets transmitted through that socket end-point, as long as the socket option is set.

However, this document provides guidelines for those implementations which like to have an option on implementing transmit-side ancillary data object support for altering default source address selection. Therefore, if an application chooses to use the per-packet source address selection, then the implementation should process at the `IPPROTO_IPV6` level (`cmsg_level`) ancillary data object of type (`cmsg_type`) `IPV6_ADDR_PREFERENCES` containing as data (`cmsg_data[]`) a 32-bits unsigned integer encoding the source address selection preference flags (e.g. `IPV6_PREFER_SRC_COA` | `IPV6_PREFER_SRC_PUBLIC`), in a fashion similar to the advanced IPv6 Socket API [[RFC3542](#)]. This address selection preference ancillary data object may be present along with other ancillary data objects.

The implementation processing the ancillary data object is responsible for selection of preferred source address as indicated in the ancillary data object. Thus, an application can use `sendmsg()` to pass an address selection preference ancillary data object to the IPv6 layer. The following example shows usage of the ancillary data API for setting address preferences:



```
void *extptr;
socklen_t extlen;
struct msghdr msg;
struct cmsghdr *cmsgptr;
int cmsglen;
struct sockaddr_in6 dest;
uint32_t flags;

extlen = sizeof(flags);
cmsglen = CMSG_SPACE(extlen);
cmsgptr = malloc(cmsglen);
cmsgptr->cmsg_len = CMSG_LEN(extlen);
cmsgptr->cmsg_level = IPPROTO_IPV6;
cmsgptr->cmsg_type = IPV6_ADDR_PREFERENCES;

extptr = CMSG_DATA(cmsgptr);

flags = IPV6_PREFER_SRC_COA;
memcpy(extptr, &flags, extlen);

msg.msg_control = cmsgptr;
msg.msg_controllen = cmsglen;

/* finish filling in msg{} */

msg.msg_name = dest;

sendmsg(s, &msg, 0);
```

Thus when an IPV6\_ADDR\_PREFERENCES ancillary data object is passed to `sendmsg()`, the value included in the object is used to specify address preference for the packet being sent by `sendmsg()`.





**Appendix B. Changes from previous version of draft**

- o Added an appendix discussing setting source address selection preferences on a per packet basis, including via passing a transmit-side ancillary data object.
- o Changed getaddrinfo() usage and removed the AI\_PREFER flags. Added one extended flag field in addrinfo structure and added one AI\_flag in order to save bits.
- o Removed scope address preferences as some questioned about applicability of IPV6\_PREFER\_SRC and DST scope rules alterations.
- o Added -1 return value for inet6\_is\_srcaddr() for failure situation.
- o Addressed comments by the working group and reviewers of this draft.
- o Added a definition of term for X and not-X flags.
- o Changed "not-X flag" by "opposite flag", and "X and not-X flags" by "contradictory flags"
- o updated rationale why not to deal with native transport destination address selection rule.



**[Appendix C](#). 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.

## Authors' Addresses

Erik Nordmark  
Sun Microsystems, Inc.  
4150 Network Circle, UMPK17-308  
Santa Clara, CA 95054  
USA

Email: Erik.Nordmark@Sun.COM

Samita Chakrabarti  
Azaire Networks  
4800 Great America Parkway  
Santa Clara, CA 95054  
USA

Email: samitac2@gmail.com

Julien Laganier  
DoCoMo Euro-Labs  
Landsbergerstrasse 312  
D-80687 Muenchen  
Germany

Email: julien.IETF@laposte.net



## Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at [ietf-ipr@ietf.org](mailto:ietf-ipr@ietf.org).

## Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

## Acknowledgment

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

