Internet Engineering Task Force
INTERNET DRAFT

Arifumi Matsumoto
Kenji Fujikawa
Yasuo Okabe
Kyoto University
23 June 2003

Basic Socket API Extensions for LIN6 End-to-End Multihoming

<draft-arifumi-lin6-multihome-api-00.txt>

Status of this Memo

Abstract

This document describes a method for multihoming support in application layer. We extend the basic socket API(Application Programming Interface) and propose some new interfaces for multihoming. Multihoming nodes are expected to have multiple addresses. The existing socket APIs, however, are not designed to manipulate multiple addresses in a connection. Proposed APIs help an application to handle multiple addresses, to avoid connection failure and to do load-balancing possibly. Right now, the proposed APIs are for LIN6 nodes, one of the mobile protocols. This is because LIN6's addressing architecture, which is called "8+8", is very friendly and consistent with multihoming. In this document, we propose a host-based multihoming solution and which is called end-to-end multihoming. In end-to-end multihoming, a fault-tolerant connection

can be achieved relying not on routers but on the pair of end-nodes only.

1. Introduction

This document describes a method of host-based multihoming, which is implemented in the application layer. We extend the basic socket interface for IPv6[RFC2553] and propose some new interfaces for multihoming. Multihoming nodes are expected to have multiple addresses. The existing socket APIs, however, are not designed to manipulate multiple addresses in a connection. Proposed APIs make multiple addresses visible to an application and help an application to handle multiple addresses, to avoid connection failure and to do load-balancing possibly.

Right now, the proposed APIs are for LIN6 nodes only. LIN6(Location Independent Network Architectur for IPv6)[LIN6], which is developed by M. Ishiyama, M. Kunishi and F. Teraoka, is one of the mobile network protocols. LIN6 is more efficient and simpler than other existing mobile network protocols are and this is mainly because of its addressing architecture. In LIN6 an IPv6 address is divided into two parts, ``locator'' and ``identifier,'' and DNS and ``Mapping Agents'' are used for the mapping of an identifier and its locators. This LIN6's addressing architecture is called "8+8", and is very friendly and consistent with multihoming. In this document, we propose a host-based multihoming solution, which is called end-to-end multihoming[E2E], combined with LIN6 mobile protocol. In end-to-end multihoming, a fault-tolerant connection can be achieved relying not on routers but on the pair of end-nodes only.

On a LIN6 node, an existing application can be executed without any modifications to the application itself. Even if a node is multihomed and has multiple addresses, however, existing applications don't benefit from having multiple up-stream access lines. With our APIs, you can make an application that actively make use of multihoming.

Although our APIs are such capable and easy to use, it is nonsense to remake all the existing applications in order to deploy multihoming. In case of TCP, the detection and avoidance of connection failure can be implemented in the transport layer by ack packet's timeout and so on. In case of UDP, however, it is nealy impossible to do the same in the transport layer. Those processes have to be done in the application layer. Thus, our APIs are mainly suitable for making multihome-ready UDP applications. Now we are planning to implement improved TCP for end-to-end multihoming.

This memo describes LIN6 protocol overview, our C language APIs

specification, an example of how to use the API in order to write an multihoming-capable applications and some notes on the deployment of this multihoming method.

2. LIN6 Protocol Overview

In conventional network architectures including IPv4 and IPv6, the network address of a node denotes both its identity and its location. In LIN6 architecture, we divide a 128bit-long IPv6 address into two parts. The first half is called ``locator'' and the second half ``identifier''. A locator only depicts a location and an identifier only depicts an identity. A LIN6 node can identify a corresponding node by examining only the second half of an IP address. This is independent of the first half, which may be changed when the node moves.

The separation of an IP address also makes it possible to support multihoming without any inconsistency. A LIN6 node located in the multihomed network has multiple global locators. Even if an network trouble occurred on one link, a corresponding LIN6 node detects it and can resume communication by using another locator and another link. In this method, a fault-tolerant connection is achieved relying only on the pair of end-nodes, not on routers. We call this method ``end-to-end multihoming.''

3. Multihoming Support for LIN6

When a LIN6 node moves and detects its movement (changing of its locator), the node sends a location update message packet to its corresponding nodes and can continue to communicate seamlessly. On the other hand, when a network trouble occurred between two nodes, the connection will be lost, even if one or both nodes are multihomed.

In such a case, a node is expected to detect network troubles by some error packets such as ICMP Host Unreach or by acknowledgement timeout. By changing destination or source locator, the traffic possibly circumvents the point of failure and hopefully the connection can be resumed. Especially when a node is located under those sites whose site exit routers perform policy routing based on the outgoing packet's source address, changing the source locate is more meaningful. In the LIN6 addressing architecture, a host is identified by only the latter half of its address, thus a connection is identified by only the latter half of the source address and the destination addresse. Therefore a LIN6 node can change a destination locator and a source locator even when a connection is established.

These processes of detecting errors and changing locators is expected to be implemented in an application or in the transport layer. The existing socket APIs, however, are not designed to manipulate multiple addresses in a connection. Thus, we design new APIs to write an multihome-ready application. Proposed APIs make multiple addresses visible to an application and help an application to handle multiple addresses, to avoid connection failure and to do load- balancing possibly.

Note that on a LIN6 node, an existing application can be executed without any modifications to the application itself. Even if a node is multihomed and has multiple addresses, however, existing applications don't benefit from having multiple up-stream access lines. With our APIs, you can make an application that actively make use of multihoming.

3.1. API Overview

The main newly designed APIs are listed below.

o socket()
  We define a new address family "AF_ALIN6" and a new protocol family "PF_ALIN6". If you specify "PF_ALIN6" for socket()'s protocol family, you will get multihome-ready socket and you can manipulate foreign and local locators through the socket. PF_ALIN6 socket is available for IPPROTO_TCP and IPPROTO_UDP.

o getaddrinfo2()
  This API returns locators of a specified corresponding node by making a query to the node's Mapping Agent. This is an extented form of the exisiting socket library function getaddrinfo(). Getaddrinfo2() performs the functionality of getaddrinfo(), name-to-address translation, and also aquires an identifier-to-locators mapping if the latter half of the IPv6 address is an LIN6 identifier. The return value is not a linked list of struct addrinfo, but a linked list of struct addrinfo2, described below, which includes target node's locators.

```
struct addrinfo2 {
  int ai_flags;    /* AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST */
  int ai_family;   /* PF_XXX */
  int ai_socktype; /* SOCK_XXX */
  struct sockaddr *ai_addr; /* binary address */
  ...
  size_t ai_ntloc; /* number of target locator */
  struct lin6_prefix* ai_tloc; /* target locators */
};
```

ai_tloc is the pointer to the first entry of the locator array.
ai_ntloc is the number of entries in the array.

o getsockopt()/setsockopt()
We define new options for getsockopt() and setsockopt(). These socket APIs are used for get and set options on sockets. We introduce new options LIN6_LOCALADDROPTS, LIN6_FOREIGNADDROPTS, LIN6_LOCALLOCATOR and LIN6_FOREIGNLOCATOR to manipulate socket behavior and to get/set the locator of the protocol control block(PCB) in the kernel. When a connection error is detected an application can try another locator by using the setsockopt() with the option LIN6_FOREIGNLOCATOR and an new locator of the target host.

Next, we show the programming example below.

3.2. Simple Example.

3.2.1 Establishment of an Connection

After getting corresponder's locators by getaddrinfo2(), this example applicaton below tries to connect to an acquired address. Getaddrinfo2() returns a pointer to a linked list of struct addrinfo2. If the address family specified in the returned struct addrinfo2 is AF_ALIN6, there may be multiple locators in it. This application tries to connect to the corresponding node using each locator until connection establishment succeeds.

```
struct addrinfo2 hints, *res, *res0;
struct sockaddr_lin6 slin6;
int error, sock;
int ntloc;

/* name to address resolution */
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
error = getaddrinfo2("mob.foo","http", &hints,&res0);
if (error) {
  errx(1, "%s", gai_strerror(error));
  /*NOTREACHED*/
}

/* look for valid address */
sock = -1;
for(res = res0; res; res = res->ai_next) {
  sock = socket(res->ai_family, res->ai_socktype,
```

```
            res->ai_protocol);
      if (sock < 0)
        continue;

      if (res->ai_family==AF_ALIN6) {
        memcpy(&slin6, res->ai_addr, sizeof(slin6));
        ntloc = ai_ntloc;
        for(i=0;i<res->ai_ntloc;i++) {
          /* try each target locator */
          bcopy(res->ai_tloc[i], slin6.slin6_locator,
            sizeof(struct locator));
          if (connect(sock, (struct sockaddr*)&slin6,
            res->ai_addrlen)==0)
            goto connected:
        }
        sock = -1;
      } else {
        ntloc = 0;
        if (connect(sock,res->ai_addr, res->ai_addrlen) < 0) {
          close(sock);
          sock = -1;
        }
        break;
      }
    }
    if (sock < 0)
      err(1);
    freeaddrinfo(res0);

    connected:
     freeaddrinfo2(res0);
    }
```

3.2.2 Connection Recovery

After the connection establishment this application may catch some error signals caused by ICMP Error Message. This application tries another locator acquired before if available and tries to continue connection.

```
  void sig_handler(int sig) {
    /* error signal handler */
    if (ntloc>0) {
      /* change locator */
      if (setsockopt(sock, IPPROTO_IPV6, LIN6_FOREIGNLOCATOR,
        ai_tloc[++i], sizeof(struct lin6_prefix)) < 0) {
        ...
```

```
        }
      }
    }
```

4. Deployment Considerations

Although our APIs are such capable and easy to use, it is nonsense to propose to remake all the existing applications in order to deploy multihoming. In case of TCP, the detection and avoidance of connection failure can be achieved in the transport layer by ack packet's timeout and so on. In case of UDP, however, it is nealy impossible to do the same in the transport layer. Those processes have to be done in the application layer. Thus, our APIs are mainly suitable for making multihome-ready UDP applications.

We have already implemented this multihoming method for UDP. Now we are planning to implement reliable transport protocol for end-to-end multihoming. This can be a modification to the existing transport protocol TCP or adoption of a brand new transport protocol, such as SCTP[RFC2960].

5. Acknowledgement

Thanks to the many people who made suggestions and provided feedback to this document, including: Fumio Teraoka, Masahiro Ishiyama, Masataka Ohta, Mitsunobu Kunishi and members and staffs in okabe laboratory and minoh laboratory.

6. References

[LIN6] M. Ishiyama, M. Kunishi, K. Uehara, H.Esaki, F. Teraoka, ``LINA: A New Approach to Mobility Support in Wide Area Networks,'' IEICE Trans. Communications, Vol E84-B, No 8, Aug 2001, pp.2076-2086.

[E2E] M. Ohta, ``The Architecture of End to End Multihoming,'' Internet-draft, IETF (Nov 2002), draft-ohta-e2e-multihoming-03.txt.

[RFC2553] R. Gilligan, S. Thomson, J. Bound, W. Stevens, ``Basic Socket Interface Extensions for IPv6 ,'' RFC2553, IETF (Mar 1999).

[RFC2960] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, ``Stream Control Transmission Protocol,`` RFC2960, IETF (Oct 2000).

7. Authors' Addresses

Arifumi Matsumoto
Graduate School of Informatics
Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501 JAPAN
Tel: +81 75-753-7468
Fax: +81 75-753-7472
Email: arifumi@net.ist.i.kyoto-u.ac.jp

Kenji Fujikawa
Graduate School of Informatics
Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501 JAPAN
Tel: +81 75-753-7468
Fax: +81 75-753-7472
Email: fujikawa@real-internet.org

Yasuo Okabe
Integrated information Network System
Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501 JAPAN
Tel: +81 75-753-7468
Fax: +81 75-753-7472
Email: okabe@i.kyoto-u.ac.jp