

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 17, 2009

L. Iannone
D. Saucez
O. Bonaventure
UCLouvain, Belgium
July 16, 2008

OpenLISP Implementation Report
draft-iannone-openlisp-implementation-01

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/lid-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 January 17, 2009.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Internet-Draft

OpenLISP Implementation Report

July 2008

Abstract

The RRG is working on the design of an alternate Internet Architecture in order solve issues of the current architecture related to scalability, mobility, multi-homing, and inter-domain routing. Among the various proposals, LISP (Locator/ID Separation Protocol) is one of the most advanced. The present draft describes the overall architecture of OpenLISP, an open source implementation of the LISP proposal. Further, the draft contains some general remarks concerning the design and the implementation of the LISP protocol.

Table of Contents

1.	Introduction	3
1.1.	Terms Definition	3
2.	Map Tables	4
3.	Protocol Stack Modifications	8
3.1.	Incoming Packets	8
3.2.	Outgoing Packets	10
3.3.	Implementation Status	12
4.	Mapping Sockets API	14
4.1.	An example of mapping sockets usage	18
5.	Sysctl API	23
6.	LISP and OpenLISP issues	24
6.1.	Multicast	24
6.2.	OpenLISP and LISP variants	24
6.3.	OpenLISP as TE-ITR/TE-ETR	24
6.4.	OpenLISP and nonce	24
6.5.	OpenLISP and RLOC order	25
6.6.	LISP Source port and statefull firewall	26
6.7.	ICMP	26
6.8.	MTU Management	27
6.8.1.	OpenLISP local MTU Management	27
6.8.2.	OpenLISP Extended MTU Management	28
7.	Conclusion	30
8.	Acknowledgements	31
9.	IANA Considerations	32
10.	Security Considerations	33
10.1.	Reachability bits DoS	33
11.	Informative References	35
Appendix A.	Man Pages	36

A.1.	map(1)	36
A.2.	map(4)	40
A.3.	mapstat	44
	Authors' Addresses	46
	Intellectual Property and Copyright Statements	47

[1.](#) Introduction

Very recent activities in the IETF and in particular in the Routing Research Group (RRG) have focused on defining a new Internet architecture, in order to solve issues related to scalability, addressing, mobility, multi-homing, inter-domain traffic engineering and routing ([\[I-D.iab-raws-report\]](#), [\[I-D.irtf-rrg-design-goals\]](#)). It is widely recognized that the approach based on the separation of the end-systems' addressing space (the identifiers) and the routing locators' space is the way to go. This separation is meant to alleviate the routing burden of the Default Free Zone (DFZ), but it implies the need of distributing and storing mappings between identifiers and locators on caches placed on routers and to perform tunneling or address translation operation.

Among the various proposals presented in various RRG's meeting, LISP (Locator/ID Separation Protocol), based on the map-and-encap approach [\[I-D.farinacci-lisp\]](#), is one of the most advanced and promising proposals. UC Louvain is currently developing an implementation, called OpenLISP, of this protocol in the FreeBSD kernel (version 7.0 - [\[FreeBSD\]](#)). OpenLISP can be downloaded from: <http://inl.info.ucl.ac.be>. Note that the current release refers to version 07 of the LISP draft.

This draft describes the overall architecture of this implementation and its main data structures. The draft is structured as follows. We first describe the kernels' data structures created to store the mappings necessary to perform encapsulation and decapsulation operations. Then, we show the architectural modifications made to the FreeBSD protocol stack in order to support the LISP protocol. Then, we describe the new mapping sockets that have been introduced in order to access the mappings from user space. This feature will be useful to develop Mapping Distribution Protocols in the user space. Finally, we discuss some issues related to the design and the implementation of the LISP proposal.

[1.1.](#) Terms Definition

The present draft uses the terms that are originally defined in [\[I-D.farinacci-lisp\]](#). For terms like EID, RLOC, ITR, ETR, etc, please refer to the original LISP specification.

[2.](#) Map Tables

LISP defines two different databases to store mappings between EID-prefixes and RLOCs. The "LISP Cache" stores short-lived mappings in an on-demand fashion when new flows start. The "LISP Database" stores all the local mappings, i.e., all the mappings of the EID-Prefixes behind the router. In OpenLISP we merged the two databases in a single radix tree data structure [\[TCPIP\]](#). This allows to have an efficient indexing structure for all the EID-Prefixes that need to be stored in the system. EID-Prefixes that are part of the LISP Database are marked by a "local" flag, indicating that they are EID-Prefixes for which the mapping is owned locally. Thus, from a logical point of view the two "databases" are still separated. Actually there are two radix structures in the system, one for IPv4 EID-Prefixes and another for IPv6 EID-Prefixes. In both map tables, each entry has the format depicted in Figure 1.

```
struct mapentry {
    struct radix_node map_nodes[2]; /* tree glue, and other values */

    struct sockaddr_storage *EID;    /* EID value */

    struct locator_chain * rlocs;    /* Set of locators */
    int    rlocs_cnt;                /* Number of rlocs */

    u_long map_flags;                /* up/down?, local */
};
```

The mapentry structure

Figure 1

Besides the fields necessary to build the radix tree itself, the entries contain a pointer to a socket address structure that holds the EID-Prefix to which the entry is related.

The "map_flags" field contains general flags that apply to the whole mapping. Insofar, four flags have been defined and are listed in Table 1. The MAPF_UP flag just states that the mapping is usable. The MAPF_LOCAL flag means that the mapping is owned locally (i.e., it is part of the LISP Database). In OpenLISP, when inserting a "local" mapping it is mandatory that at least one RLOC is a local address; i.e., an address of one of the interfaces of the system, otherwise, during insertion, the system will return EINVAL error. This is because, when OpenLISP performs encapsulation, it only selects source RLOCs that are addresses of the system. Doing otherwise would

introduce the risk of packet filtering on upstream routers if packets are sent with a source address that does not belong to the system performing the encapsulation operation. The MAPF_STATIC indicates that the mapping has been manually added, e.g., through the map utility (see [Appendix A.1](#)). The MAPF_DONE flag is used for messages through mapping sockets (see [Section 4](#)). Note that in the actual release of OpenLISP, both static and non-static entries are treated in the same way: they need to be explicitly deleted. Future releases of OpenLISP will include the possibility to introduce a timeout for non-local entries.

Constant	Value	Description
MAPF_UP	0x1	Mapping usable.
MAPF_LOCAL	0x2	Mapping is local. This means that it should be considered as part of the LISP Database.
MAPF_STATIC	0x4	Mapping manually added.

MAPF_DONE	0x8	Message confirmed.
-----------	-----	--------------------

Table 1: General mapping flags

The other main field of the mapentry data structure is the `rlocs_cnt` field, containing the number of RLOCs present in the mapping. These RLOCs are stored in a chained list whose head is referenced by the "rlocs" pointer. The list of RLOCs is always maintained ordered by increasing priority values, which means that RLOCs with higher priority are at the head of the list.

Each element of the RLOCs list is a socket address structure containing the locator and an `rloc_mtx` structure. The latter, depicted in Figure 2, contains the priority and weight parameters, whose meaning and use are defined in the original LISP specification (including the particular 255 value for the priority field). Note that load balancing is not yet implemented in OpenLISP, thus the weight is not considered during RLOC selection. Future versions of OpenLISP will include load balancing and hence full support of the weight parameter. Furthermore, there is also a flags field, for flags that are specific to a RLOC, and a `mtu` field.

```

struct rloc_mtx { /* Metrics associated to the RLOC
                  */
    u_int8_t priority; /* Each RLOC has a priority.
                       * A value of 255 means that
                       * RLOC MUST not be used.
                       */
    u_int8_t weight; /* Each locator has a weight.
                    * Used for load balancing
                    * purposes when two or more
                    * locators have the same
                    * priority.
                    */
    u_int16_t flags; /* RLOC-related flags.
                    */

```

```

        u_int32_t mtu;          /* MTU for the specific RLOC.
                                */
};

```

RLOCs metric data structure.

Figure 2

Constant	Value	Description
RLOCF_REACH	0x1	RLOC Reachable.
RLOCF_LIF	0x2	RLOC is a local address. This valid only for mappings with the MAPF_LOCAL flag set.

Table 2: RLOC Specific flags

Concerning flags, there are only two RLOC specific flags defined insofar and described in Table 2. The RLOCF_REACH flag just indicates if the RLOC is reachable or not. This flag is meaningful no matter if the mapping is local or not. The RLOCF_LIF flag is meaningful only for local mappings and indicates if the RLOC address belongs to the system. When performing encapsulation, a RLOC is selected from a local mapping only if it has this flag set, it is reachable, and its priority is less than 255. This in order to issue packets that have a source address which belongs to the system itself.

The "mtu" field is used to check if the size of the LISP-encapsulated packet fits the MTU (Maximum Transmission Unit) of the outgoing

interface. OpenLISP automatically fill this field when a local mapping is added. In particular, OpenLISP checks all the RLOCs of the local mappings, if it is an address belonging to the system it sets the RLOCF_LIF flag and copies the MTU of the interface associated to the address. Note that, the check is done only upon insertion, thus changes in the local address or the MTU are not automatically copied in the mapping entry. For details on the use of this field please refer to [Section 6.8](#).

The use in OpenLISP of a chained list to store the RLOCs, allows mixing IPv4 and IPv6 RLOCs. This in turn allows to use IPv6 tunneling for IPv4 packets and vice versa. Even more, in this way it is possible, for the same EID, to perform both IPv6 and IPv4 tunneling depending on the RLOC eventually chosen for the encapsulation. This avoids the constraint of having the tunnels toward the same EID either all IPv4 or all IPv6. Even if in the actual implementation status of OpenLISP, both IPv4 and IPv6 EIDs mapping tables are present, and both IPv4 and IPv6 RLOCs can be introduced without limitation on the EID address family, the encapsulation and decapsulation operation are implemented only for IPv4, as long as the map and mapstat utility (see [Appendix A.1](#) and [Appendix A.3](#)). Future releases of OpenLISP will support IPv6 encapsulation.

[3.](#) Protocol Stack Modifications

Compared to the original protocol stack implementation of the FreeBSD OS ([\[TCPIP\]](#), [\[FreeBSD\]](#)) four main modules have been added, namely `lisp_input()`, `lisp6_input()`, `lisp_output()`, and `lisp6_output()`. As should be clear from the names, the first two modules manage incoming IPv4 and IPv6 LISP packets, while the last two modules are responsible for outgoing IPv4 and IPv6 LISP packets. To describe the global architecture, we use the same module representation as in [\[TCPIP\]](#) and show how packets are processed inside the protocol stack.

[3.1.](#) Incoming Packets

The `lisp_input()` and `lisp6_input()` modules are positioned right above respectively the `ip_input()` and `ip6_input()` modules, from which they are called, as depicted in Figure 3.

Let us for simplicity assume that an IPv4 LISP packet is received by the system. The packet will be first treated by the `ip_input()` module. The `ip_input()` module has been patched in order to recognize LISP packets. The patch consists simply to divert towards `lisp_input()`, all incoming UDP packets destined to the local machine and having destination port number set to the LISP reserved value 4341 (for encapsulated data packets). If the UDP packet has not such a port number it is delivered as usual to the transport layer (i.e., `udp_input()`). In the case of an encapsulated data packet (port number 4341), the module strips the UDP header and then it treats the reachability bits and the nonce of the LISP specific header.

OpenLISP checks all of the reachability bits and updates reachability information in the map tables. While performing such an update a consistency check is performed. In particular, the number of reachability blocks (32 bits) present in the packet is compared to the number of reachability blocks present in the matching mapping entry, if different the packet is dropped for bad LISP encapsulation and a message is sent through open mapping sockets (see [Section 4](#)).

Protocol Stack Modifications for incoming packets.

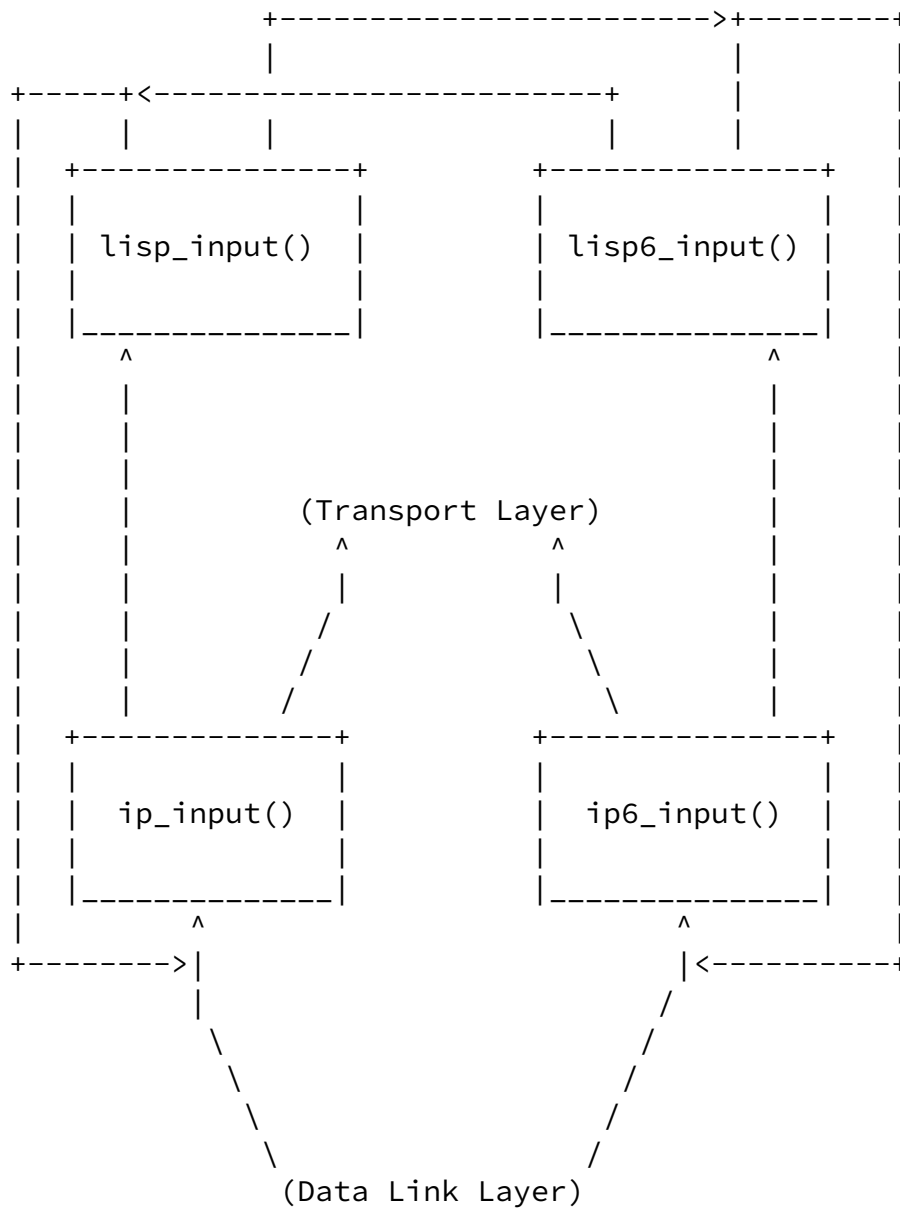


Figure 3

The same action is triggered if the number of reachability bits that are set in the blocks differs from what expected. For instance, let us assume the stored mapping contains two (2) RLOCs and the incoming packets contains three (3) reachability bits set, this should never happen, thus the packet is discarded and a message is sent through open mapping sockets, in order to inform possible existing mapping management processes. Note that a mismatch in the number of reachability bits can be discovered only if they are set to reachable. If everything matches, but some reachability bits have

changed, they are updated in the mapping and a message is sent through open mapping sockets to notify this change. Note that this

last message is not a an error message, but just a notification message, necessary to notify possible mapping management processes in the user space about the reachability change.

After having performed these operations, the IP header of the remaining packet is checked in order to decide to which module to deliver the packet. In practice this means to re-inject the packet in the IP protocol stack, by putting it in the input buffer either of the `ip_input()` or the `ip6_input()` module.

In the case of an IPv6 LISP packet the overall process is the same. The packet is first received by `ip6_input()`, where if the packet is a locally destined UDP packet with destination port number equal to the LISP reserved 4341 value it is delivered to `lisp6_input()`. The latter module performs the same operations as `lisp_input()`, with the only difference that it is specialized in treating IPv6 headers. If the packet is a data packet, depending on the address family of the inner header, once decapsulated it is re-injected either in the input buffer of the `ip_input()` module or the input buffer of `ip6_input()` module.

Once the packet is re-injected in the protocol stack, in both IPv4 and IPv6 cases, the packet follows the normal process. This means that if the decapsulated packet is not destined to the local host it will be first delivered to the forwarding module (`ip_forward()` or `ip6_forward()`) that will in turn deliver it to the output module (`ip_output()` or `ip6_output()`) in order to send it down to the data link layer and transmit it toward its final destination. These last actions are driven by the content of the routing table of the system.

[3.2.](#) Outgoing Packets

The `lisp_output()` and `lisp6_output()` modules are positioned right above respectively the `ip_output()` and `ip6_output()` modules, from which they are called, as depicted in Figure 4.

Protocol Stack Modifications for outgoing packets.

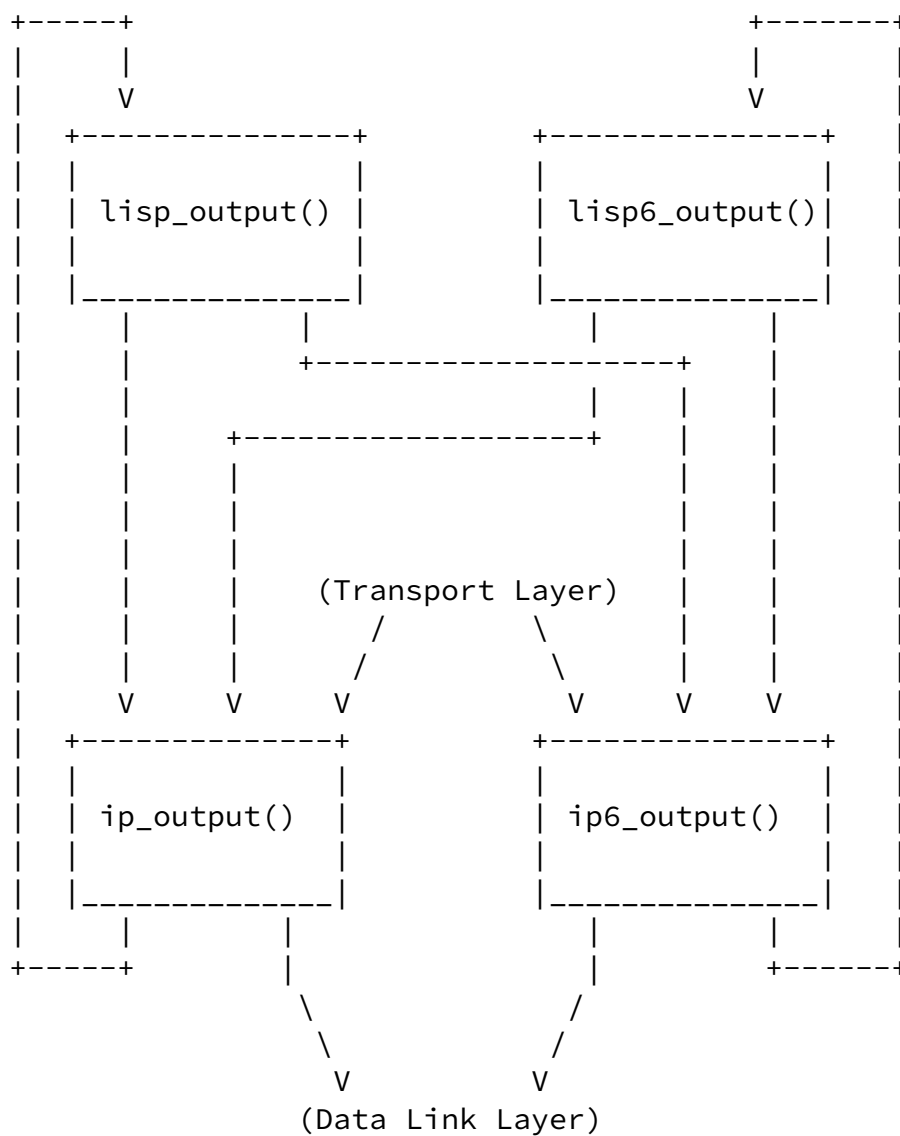


Figure 4

Let us for simplicity assume that an IPv4 is received by the `ip_output()` module, coming either from the `ip_forward()` module or the transport layer (i.e., either `tcp_output()` or `udp_output()`). Note that we refer to a normal IPv4 packet, not a LISP encapsulated packet. The `ip_output()` module has been patched in order to recognize if the packet needs to be encapsulated with a LISP header. The patch consists in first checking if there is a valid mapping in the LISP database. This means to perform a search in the map table using the source address (source EID) of the packet. If the lookup returns an entry with the `MAPF_LOCAL` flag set (recall [Section 2](#)), then a second lookup is performed in order to find a mapping for the destination EID.

If there is no mapping available a message is sent through open mapping sockets in order to notify the cache miss. This is needed in order to trigger a mapping lookup, i.e., to send a Map-Request, by the Mapping Distribution Protocol. Since there is no mapping available the packet is not encapsulated. It is normally treated by the IP layer, which means that if the destination EID is routable and a route exist in the IP routing table it is forwarded without being encapsulated. Otherwise the IP layer will drop it.

If a mapping for the destination EID is present, the packet is diverted toward the `lisp_output()` module. The `lisp_output()`, will first perform MTUs checks (see [Section 6.8.1](#)), then it prepends to the packet the LISP header (i.e. reach bits and nonce). Final step is to prepend a new IP + UDP header using selected RLOCs. The destination RLOC is selected using the policy described in the original LISP specification. The source RLOC is chosen in a slightly more restrictive way, as described in [Section 2](#).

Subsequently the packet is sent again to the IP layer in order to ship it to the data-link layer. This does not mean that the packet is delivered to `ip_output()`. Indeed, the mapping for the destination address can have an IPv6 RLOC as a first element of the list of locators, meaning that the prepended header is IPv6+UDP and that the packet is delivered to the `ip6_output()` module. Note that the new LISP encapsulated packet cannot be recursively encapsulated. Indeed,

the mbuf containing the packet is tagged with a new M_TAG_LISP tag, which avoids to re-perform encapsulation check by performing lookups on the map tables. This allows to reduce computational overhead while protecting against bad setups generating loops where a packet is recursively encapsulated until it is dropped due to MTU checks.

In the case of an outgoing IPv6 packet the overall process is the same. The packet, if a mapping exists for the source EID, is first diverted toward `lisp6_output()`, which prepends the correct headers to the packet and, depending of the RLOC used, delivers the packet either to the `ip_output()` module or the `ip6_output()` module.

Once the packet is re-injected in the protocol stack, in both IPv4 and IPv6 cases, the packet follows the normal process. This means that the encapsulated packet will be delivered to the data-link layer.

[3.3.](#) Implementation Status

In the current public release of OpenLISP, only the modules `lisp_input()` and `lisp_output()` are present. Thus only IPv4 encapsulation/decapsulation operations are supported. Future releases will include encapsulation/decapsulation support for IPv6.

Similarly, the gleaning mechanism is not yet supported. Thus, OpenLISP is not able to generate and manage Data-Probe packets.

4. Mapping Sockets API

In line with the UNIX philosophy and to give the possibility for future Mapping Distribution Systems running in the user space to access the kernel's map tables a new type of socket, namely the "mapping sockets", has been defined.

Mapping sockets are based on raw sockets in the new AF_MAP domain and are very similar to the well known routing sockets ([[TCPIP](#)], [[NetProg](#)].) A mapping socket is easily created in the following way:

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/mappings.h>
```

```
int s = socket(PF_MAP, SOCK_RAW, 0);
```

Note that `<net/mappings.h>` is the header file containing all the useful data structures and definitions.

Once a process has created a mapping socket, it can perform the following operations by sending messages across it:

MAPM_ADD: used to add a mapping. The process writes the new mapping to the kernel and reads the result of the operation on the same socket.

MAPM_DELETE: used to delete a mapping. It works in the same way as MAPM_ADD.

MAPM_GET: used to retrieve a mapping. The process writes on the socket the request of a mapping for a specific EID and reads on the same socket the result of the query.

The messages sent across mapping socket for the above operations all use the same data structure, namely `map_msghdr{}`, depicted in Figure 6.

The field `map_type` can be set only to the type listed above. The fields `map_msglen`, `map_version`, `map_pid`, `map_seq`, and `map_errno` have the same meaning and are used in the same way as for the `rt_msghdr{}` structure for routing sockets. Details about these fields and their use can be found in [\[TCPIP\]](#). The `map_flags` field is used to set some general flags that concern the whole mapping entry or the message. The possible values are listed in Table 1 along with their meaning in

[Section 2](#). The only value that was not described in [Section 2](#) is the MAPF_DONE flag. This particular flag is set by the kernel and just state that the operation requested has been performed successfully. Note that all of the messages are returned by the kernel and copies

are sent to all interested listeners (open mapping sockets). A process may avoid the expense of reading replies to its own messages by issuing a `setsockopt(2)` call indicating that the `SO_USELOOPBACK` option at the `SOL_SOCKET` level is to be turned off. A process may ignore all messages from the mapping socket by doing a `shutdown(2)` system call for further input.

Mapping Message Header.

```
struct map_msghdr {          /* From maptables.h
                               */
    u_short map_msglen;      /* to skip over non-understood
                               * messages
                               */
    u_char  map_version;     /* future binary compatibility
                               */
    u_char  map_type;        /* message type */
    int     map_flags;       /* flags, incl. kern & message,
                               * e.g. DONE
                               */
    int     map_addrs;       /* bitmask identifying sockaddrs
                               * in msg
                               */
    int     map_rloc_count;  /* Number of rlocs appended to
                               the msg */
    pid_t   map_pid;        /* identify sender
                               */
    int     map_seq;        /* for sender to identify action
                               */
    int     map_errno;       /* why failed
                               */
};
```

Figure 6

When trying to install a new mapping, the OpenLISP code can return the following error codes if something goes wrong:

ENOBUFFS: If insufficient resources were available to install a new mapping.

EEXIST: If the EID-Prefix already exists in the mapping table.

EINVAL: This error code can be returned in two cases. The first case is when the list of RLOC provided for a mapping contains replicated addresses. The second case is when a "local" mapping is provided without any RLOC (address) belonging to the system. Note the OpenLISP does not support Negative Mapping Entries.

As can be noted, the use of the MAPF_LOCAL flag allows to use the mapping socket API for mappings in both the LISP Database and LISP Cache. As explained in [Section 2](#), they are merged in the radix data structure in order to have an efficient lookup mechanism for all possible EIDs.

The OpenLISP kernel code can trigger some messages to be sent through the mapping sockets if some particular events take place. The messages triggered by the kernel are the following:

MAPM_MISS: a lookup operation has generated a miss (mapping not present). This message is generated when a LISP encapsulated packet is received, but no mapping exists, in the map tables, for the source EID.

MAPM_BADREACH: a LISP encapsulated packet has been received but the reachability bits do not match existing mapping. This message informs possible existing mapping distribution systems in the user space that a non recoverable mismatch has been detected between the reachability bits in the header of a LISP encapsulated packet and what expected from the mapping present in the map tables. Details on this case can be found in [Section 3.1](#).

MAPM_REACH: reachability bits have changed. This message informs possible existing mapping distribution systems in the user space that reachability bits in an existing mapping have changed due to the reception of an LISP encapsulated packet.

Note that the above messages contain the EID for which the message has been triggered. On the one hand, this allows interested existing mapping distribution systems, in case of a MAPM_REACH message, to retrieve the updated mapping by means of a MAPM_GET message. On the other hand, for both MAPM_REACH and MAPM_BADREACH messages, the mapping distribution system in the user space can issue a Map-Request message in order to either ask confirmation of the change to the mapping owner or to obtain a fresh mapping.

The complete list of possible mapping sockets messages and their type

values are summarized in Table 3.

Constant	Value	Description
MAPM_ADD	0x01	Add Map.
MAPM_DELETE	0x02	Delete Map.
MAPM_GET	0x04	Returns mapping for a specific EID.
MAPM_MISS	0x05	Lookup Failed.
MAPM_BADREACH	0x06	Reachability Bits Problem.
MAPM_REACH	0x07	Reachability Bits Changed.

Table 3: Mapping Socket Message Types

Constant	Value	Description
MAPA_EID	0x1	EID socket address present.
MAPA_EIDMASK	0x2	EID netmask socket address present.
MAPA_RLOC	0x4	At least one RLOC is present. The exact number of RLOCs can be found in the map_rloc_count field.

Table 4: Data structure bitmask

The map_addrs field is a bitmask identifying the nature and number of data structures present in the message right after the header. The possible values and related descriptions can be found in Table 4.

The map_addrs field does not contain exactly all the data structures, in particular, for RLOCs, a bit just states if at least one RLOC is present. The exact number of RLOCs present is contained in the map_rloc_count field. While EID and its mask, if present, are simple

socket address structures, an RLOC is composed of a socket address structure followed by an `rloc_mtx` structure containing the metrics of that specific RLOC. The `rloc_mtx` data structure has been described in [Section 2](#), and is depicted in Figure 2 with a description of each metric.

[4.1](#). An example of mapping sockets usage

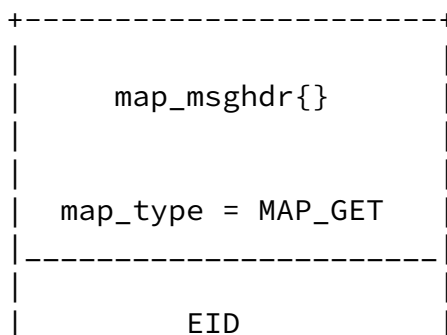
Hereafter is described an example using mapping sockets. Along with the code in the kernel, a small utility called "map" has been written. This utility has similar functionalities to the "route" utility present in UNIX systems. It allows to manually manage map tables. The complete man page of the map utility can be found in [Appendix A.1](#).

Assuming we want to retrieve the mapping for the EID 10.0.0.1, we can type:

```
freebsd% map get -inet 10.0.0.1
```

The map utility first builds a buffer containing a `map_msghdr{}` structure, followed by a socket address structure containing the EID for the kernel to look up, as depicted in Figure 8. The `map_type` is set to `MAPM_GET` and the `map_addr` is set to `MAPA_EID`. The entire buffer is written to a mapping socket previously open.

Data sent to the kernel across mapping socket for `MAP_GET` command.



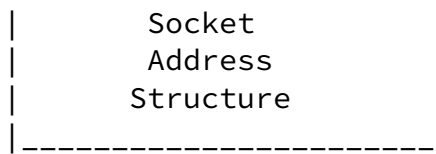
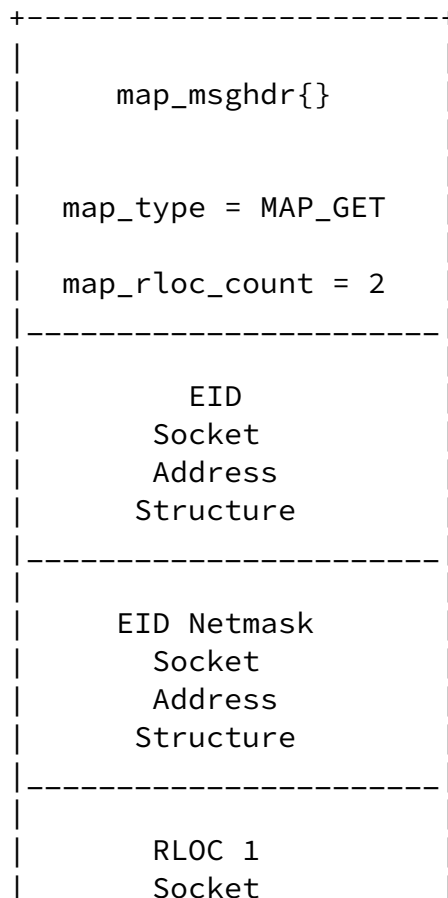


Figure 8

Afterwards, map reads from the socket the reply of the kernel. Assuming that the kernel has a mapping for 10.0.0.0/16 associated to two locators, the kernel will reply with a message which has the format depicted in Figure 9.

Data sent from the kernel across mapping socket for MAP_GET command.



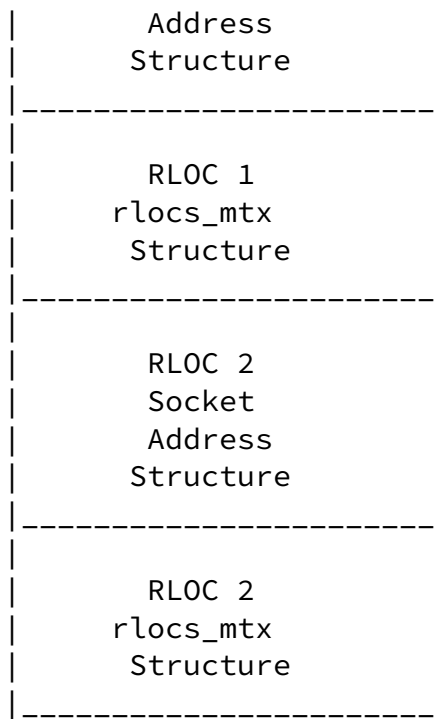


Figure 9

The first part of the message is a `map_msghdr{}` structure, with the `map_type` unchanged, the `map_addrs` set to `0x07`, which is equivalent to `MAPA_EID`, `MAPA_EIDMASK`, and `MAPA_RLOC` all set, and finally the `map_rloc_count` set to 2. Right after the `map_msghdr{}` there is a first socket address structure containing the EID prefix, which is `10.0.0.0` in this example. The second socket address structure contains the netmask, `255.255.0.0` in this case. The third socket address structure contains the first RLOC. RLOCs are returned ordered by increasing priority. After the first RLOC there is an `rloc_mtx` structure containing the metrics associated to the first RLOC. The message ends with the socket address structure for the second RLOC and the `rloc_mtx` structure for its metrics.

When using the `map` utility a possible output for the `get` request for EID `10.0.0.1` can be:

```
freebsd% map get -inet 10.0.0.1
Mapping for EID: 10.0.0.1
EID: 10.0.0.0
```

```
EID mask: 255.255.0.0
RLOC Addr: inet6 2001::1      P 1   W 100 Flags R   MTU 0
RLOC Addr: inet  10.1.0.0     P 2   W 100 Flags      MTU 0
flags: <UP,STATIC,DONE>
```

The above output is of straightforward reading. The requested lookup for EID 10.0.0.1 matches the entry with EID address 10.0.0.0 and EID mask 255.255.0.0 (/16). Note that since the map tables are radix trees, the longest prefix match is always returned. The mapping contains two (2) RLOCs. The first is the IPv6 RLOC 2001::1, having priority equal to 1, weight equal to 100, the R flag indicates that the RLOC is reachable, the MTU equal to 0 just states that no MTU is actually set. The second RLOC, is the IPv4 RLOC 10.1.0.0, having priority equal to 2, weight equal to 100, it has no flags, thus it is not reachable, and the MTU is not set.

Using the map utility, the command line to set the above-described mapping is:

```
freebsd% map add -inet 10.0.0.0/16 -inet6 2001::1 1 100 1
                                     -inet 10.1.0.0 2 100 0
```

Further examples of the map utility can be found in [Appendix A.1](#). A useful exercise in order to get familiar with the content of mapping socket messages is to run the map utility in "monitor" mode in one terminal, by typing:

```
freebsd% map monitor
```

while modifying the mapping tables using the map utility in another terminal. The monitor mode of the map utility just dumps all the messages going through mapping sockets.

Along with the map utility the "mapstat" utility is provided with OpenLISP. Mapstat is a modification of the netstat utility, already present on FreeBSD, able to provide LISP specific information. In particular a new "-X" option has been added in order to obtain a dump of the map tables. Referring to the mapping previously described,

the result of the mapstat utility would be:

```
freebsd% mapstat -X
Mapping tables
```

```
Internet:
```

EID	Flags	Refs	#	RLOC(s)				
10.0.0.0/16	US	1	1	2001::1	1	100	R	0
			2	10.1.0.0	2	100		0
								34
								43

The dump shows how only the 10.0/16 mapping is present in the map tables. The general flags show that the mapping is up ("U") and static "S", one reference exists to this mapping. Then there are the RLOCs. The information for the two RLOCs is the same like for the get command of the map utility, except for two differences. The first difference is the "#" column, which shows the position of the RLOC in the chained list of RLOCs. Second difference is the last value of the line: it expresses the number of time the RLOC has been selected for an encapsulation operation.

Along with the "-X" option, mapstat can show LISP-related network status. Where applies, mapstat accepts also the word "lisp" as protocol. As an example the following command:

```
freebsd% mapstat -sf inet -p lisp
```

will give the following result:

```
freebsd% mapstat -sf inet -p lisp
lisp:
```

```
0 datagrams received
0 with incomplete header
0 with bad encap header
```


- 0 with bad data length field
- 0 delivered
- 0 datagrams output
- 0 dropped on output
- 0 sent

The first five (5) counters concern incoming LISP encapsulated packets. In particular, the first counter gives the total number of LISP encapsulated packets received by the system. The following three gives the number of LISP encapsulated received packets dropped due to header problems or data length field problem. The fifth counter expresses the total number of packets correctly decapsulated and handed back to the IP layer.

The remaining three (3) counters concern packet received by the OpenLISP module which have a mapping for both source and destination EID and thus need to be encapsulated. The first of these three counters expresses the total packets received for encapsulation by the OpenLISP module. The second counter gives the number of packet dropped due to error conditions. The last counter gives the total number of LISP encapsulated packets that have been correctly sent.

For further information on the mapstat utility please refer to [Appendix A.3](#)

5. Sysctl API

OpenLISP offer the possibility to mapping distribution system in the user space to obtain a complete dump of the map tables through sequence of mapping messages. This is done by using a sysctl system call in the CTL_NET level. For details on the general sysctl API and its levels, in the FreeBSD systems, please refer to sysctl(3) man page. With OpenLISP is possible to use AF_MAP as second level and NET_MAPTBL_DUMP as fifth level. The sequence of messages returned by the system call is the same described in [Section 4](#).

An example of sysctl usage to obtain a map tables' dump is the following:

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <net/if.h>
#include <sys/sysctl.h>
#include <net/maptables.h>

int mib[6];
size_t spaceneeded;
char * buffer;

mib[0] = CTL_NET;
mib[1] = PF_MAP;
mib[2] = 0;
mib[3] = 0;
mib[4] = NET_MAPTBL_DUMP;
mib[5] = 0;

if (sysctl(mib, 6, NULL, &spaceneeded, NULL, 0) < 0)
    /* code for error handling */

if ((buffer = malloc(needed)) == NULL)
    /* code for error handling */

if (sysctl(mib, 6, buffer, &needed, NULL, 0) < 0)
    /* code for error handling */
```

At the end of this code, the memory buffer referenced by the "buffer" pointer will contain a contiguous sequence of mapping messages, all of them starting with the map_msghdr{} data structure, thus it can be easily parsed.

[6.](#) LISP and OpenLISP issues

In this section, we briefly discuss several of the protocol/ implementations issues/status related to OpenLISP and LISP.

[6.1.](#) Multicast

OpenLISP has no support for multicast. Future release of OpenLISP may introduce support for it.

[6.2.](#) OpenLISP and LISP variants

OpenLISP does not implement any EID filtering policy, while adopting a fallback strategy for encapsulation. This means that packets for which there is no mapping available are handed back to the IP layer for "traditional" processing. If the original packet has a non-routable destination address it will be dropped, otherwise, if a route is available, it will be forwarded. This means that OpenLISP is able, with the correct mappings to support all variants of LISP described in [[I-D.farinacci-lisp](#)].

[6.3.](#) OpenLISP as TE-ITR/TE-ETR

The lack of EID filtering policies in OpenLISP allows it to be used as also as TE-ITR/TE-ETR. The correct functioning is just a matter of putting the correct mappings in the map tables. Nevertheless, note that recursive encapsulation cannot be done on the same machine. In order to avoid inner loops (see [Section 3.2](#)), each packet once encapsulated is tagged and never checked again for further encapsulation.

[6.4.](#) OpenLISP and nonce

The original proposal of LISP includes a "nonce" value to be included in every LISP encapsulated packet. Formal definition is:

LISP Nonce: is a 32-bit value that is randomly generated by an ITR. It is used to test route-returnability when an ETR echoes back the nonce in a Map-Reply message.

In the current OpenLISP implementation, the nonce is generated and put in the LISP header, but its value is never checked on reception

of a LISP encapsulated packet. This is because the current release of OpenLISP does not support Map-Reply packets. Nevertheless, this let us think that marking every packet with a nonce is not strictly necessary, rather it introduces useless overhead. Moreover, the random generation of the nonce can be also expensive in terms of encapsulation operation performances.

In order to reduce the overhead it would be desirable to avoid putting the nonce in normal LISP encapsulated packets. Nonce can be introduced in packets that really need the value present, which are easy to recognize.

Indeed, a Map-Reply message is sent only in two cases:

- 1 To reply to an explicit Map-Request message.
- 2 In the case of the gleaning mechanism, to reply to a Data-Probe packet.

In the first case, messages are generated in the user space using as port number the IANA reserved value 4342. In this case, it is easy to recognize LISP signaling packets (Map-Request and Map-Reply) since they use destination port 4342, and thus nonce value can be handled.

In the second case, the Data-Probe should contain the nonce value in order to provide the value that needs to be returned by the subsequent Data-Reply. Data-Probe packets use destination port 4341, the same as normal LISP encapsulated data packets. However, Data-Probe packets are easily recognizable by the fact that the inner IP header and outer IP header contain the same destination address. Thus nonce can be correctly handled.

To summarize, the suggestion here is to avoid in general the nonce in normal LISP encapsulated packets, while use it in Data-Probe, Map-Request, and Map-Reply packets, which are easy to recognize. This means to split the packets' format in two main types: with and without nonce. This would allow reducing overhead in both terms of bandwidth and efficiency in the encap/decap operations.

[6.5](#). OpenLISP and RLOC order

The LISP specification clearly state that RLOCs are ordered by

priority, however, it does not clarify what happens in the case of multiple RLOCs having the same priority value. The ordering of RLOC is very important since it is used in the reachability bits.

OpenLISP uses the simple approach of considering the IP address of RLOCs (in network byte order) as an integer value and puts smaller values before bigger ones. When RLOCs belong to different address family, i.e., IPv4 and IPv6, IPv4 (AF_INET) address family is given priority. Since in OpenLISP duplicated RLOCs for the same EID-Prefix are not allowed this gives a strict ordering to the list of RLOCs.

[6.6.](#) LISP Source port and statefull firewall

During our tests with OpenLISP, we observed packet losses on high load traffic on a network protected by an IPFW statefull firewall. These packet losses were caused by the utilization of random UDP source ports for LISP packets. In [[I-D.farinacci-lisp](#)], Section 5.3, there is the following statement:

UDP Header: contains a random source port allocated by the ITR when encapsulating a packet. The destination port MUST be set to the well-known IANA assigned port value 4341.

This can be interpreted in two ways:

- o In each packet we put a random source port number. This has proved not to work well with the "keep-state" directive of IPFW. Loss of packets has been observed on high load traffic on a network protected by an IPFW statefull firewall. On statefull firewall, a state is kept for each flow, which is identified by source and destination IP addresses and source and destination port number. In presence of many different flows (due to random source port selection), the number of cached tuples (Source IP, Destination IP, Source Port, Source Destination, Protocol) can fill the firewall cache and block any new flow for a period of at least the time an entry remains in the firewall state. The random selection of UDP source ports caused a kind of DoS attack against the state maintained by the statefull firewall.

- o For the first packet to a certain RLOC we select a port number and use it as long as the mapping is valid. This is not much meaningful, since LISP never uses the source port for a reply or something else, thus this state is wasteful.

For the above reasons, in OpenLISP, the LISPDATA (4341) port number is used for source port for all LISP encapsulated packets.

6.7. ICMP

In LISP, it is not possible to find the actual source of a packet responsible of an ICMP if it occurs during the transit (i.e., when the packet is encapsulated in a LISP message).

The problem comes from the encapsulation. The returned ICMP message has sufficient space only to include the outer header, thus the one containing RLOCs as source and destination addresses. In this way it is not possible to forward the packet to the source of the original packet, since it is not possible to retrieve the original source EID. Even performing a lookup on the LISP database, using the source RLOC

as search key, the result will be an EID-Prefix, not sufficient to forward the packet.

A solution would be to increase the size of ICMP messages in order to include the inner header of the LISP encapsulated packet. This would allow to retrieve the correct information in order to forward the packet. Note, however, that before forwarding the ICMP packet needs to be cleaned from LISP specific information, since end-system are supposed to be unaware of being behind a LISP router. On the one hand, this proposition seems to be the efficient, but needs to modify ICMP and thus non-LISP routers. On the other hand, many routers that do not generate ICMP messages, or rate limit them, in the DFZ, thus reducing the real effectiveness of the solution.

For the above mentioned reasons, OpenLISP does not implement any technique that allows the router to make a link between the LISP packet header the packet source in order to "translate" and re-route ICMP packets.

The general ICMP problem in LISP can however be the pretext of a more philosophical discussion. Indeed, as LISP is a tunneling technique

based on the separation of ID and locator space, is it required to send information about what happens between the RLOCs to the client running behind a LISP router? In principle the answer is no, since end-systems do not need to be aware of the routing infrastructure (i.e., the RLOC space). In this case LISP has to handle ICMP in a different way that needs to be explored.

[6.8.](#) MTU Management

In the present section we describe how OpenLISP deals with the MTU issue inside the local domain, and how this approach can be easily extended to solve the issue on an Internet scale, without modifying existing ICMP messages.

[6.8.1.](#) OpenLISP local MTU Management

During preliminary tests, we observed that the MTU issue is at the origin of many problems. OpenLISP does not (and will not) implement the fragmentation mechanism proposed in Sec. 5.4 of [\[I-D.farinacci-lisp\]](#). The reason is because the proposed method sounds very primitive and does not appear to be efficient. The original LISP specification is based on an architectural constant used by the xTR to limit the MTU of LISP encapsulated packets. OpenLISP uses a more advanced solution, based on the real MTU of the local RLOCs present on the xTR, as described below.

Currently OpenLISP manages the MTU issue in the following manner. As

described in [Section 2](#) for each local mapping, OpenLISP discovers the RLOCs that are local interfaces and copies the MTU associated to the interface to the RLOC entry. When a packet needs to be encapsulated the first step is to calculate the final packet size and compare it to the MTU contained in the source RLOC used. If the size exceeds the MTU the action taken depends on the origin of the packet. If the packet has been locally generated through a socket in the user space, the write operation on the socket will return an EMSGSIZE error. If the packet has been originated elsewhere, an ICMP Too Big message is sent back to the source address of the original packet. Note that this can be done since the size check is done before actually encapsulating the packet.

[6.8.2.](#) OpenLISP Extended MTU Management

The way OpenLISP manages MTU solves the problem only for the local domain and the first hop after the ITR. It does not yet solve the issue of having an ICMP Too Big message generated in the middle of the LISP tunnel. A possible solution could be the enlargement of the ICMP Too Big message, as described in [Section 6.7](#).

Another possible solution is to start using the mtu field in the rloc_mtx structure also for non-local mappings. In the current OpenLISP implementation, the mtu field for RLOCs of non-local mapping are set to zero (0), which means to ignore it. If an ICMP Too Big Message is triggered in the middle of a LISP tunnel, it will normally reach the ITR that has performed the encapsulation and its content is sufficient to retrieve the destination RLOC toward which the packet was sent. This in turns allows setting a MTU on the RLOC of the mapping entry containing it. This would allow perform a check on the subsequent packets before encapsulating them, and if necessary, to send an ICMP Too Big message back to the real source of the packet.

From an architectural perspective, the proposed approach is very simple. Nevertheless, a limitation can be found in the fact that the approach suffers from some delay. Indeed, for an ICMP Too Big message to reach the original packet source, two large packets are needed. The first packet will trigger an ICMP message in the LISP tunnel, thus updating the ITR. Only the second packet will trigger an ICMP message from the ITR to the source, making the latter shrink its path MTU. However, this solution still needs to be carefully explored, since a burst of large packets must not have the results of generating a burst of ICMP messages reducing too much the MTU size on the ITR. A simple rate limitation approach can help in alleviating this problem.

A second limitation of this approach can be found in the fact that in order to rapidly update the mapping when an ICMP Too Big message is

received from a LISP tunnel, an RLOC-based lookup should be performed. In the current state of OpenLISP, this is not possible, since the mapping tables are radix trees using EIDs as key. On the other hand, RLOC-based lookup will not be that common (compared to the number of EID-based lookups), the trade-off between lookup efficiency and data structure complexity needs to be further explored.

Note, finally, that MTU discovery between RLOCs can be also performed using proposals like [[I-D.templin-seal](#)] or [[I-D.van-beijnum-multi-mtu](#)], and adapting them in order to put the correct value in the mtu field associated to RLOCs in the OpenLISP implementation. Such an adaptation is out of the scope of OpenLISP, even if worth to be explored.

7. Conclusion

The present memo describes the overall architecture and the implementation status of OpenLISP, an implementation of the LISP proposal in the FreeBSD OS. OpenLISP provides support for encap/decap operations and EID-to-RLLOC mappings storage in the kernel space. OpenLISP is freely available at <http://inl.info.ucl.ac.be>.

OpenLISP can work as both a router and end-host, thus providing a wide range of test scenarios. We think that the mapping sockets introduced by OpenLISP is a great tool for easy development of Mapping Distribution Protocols in the user space. People working in this area can contact authors. We believe that a complete working system composed by OpenLISP and a mapping distribution protocol would provide very helpful insights, leading to important improvements for both OpenLISP and the mapping distribution protocol.

[8.](#) Acknowledgements

The work described in the present memo has been partially supported by the European Commission within the IST AGAVE Project and a Cisco URP grant.

[9.](#) IANA Considerations

This memo includes no request to IANA.

[10.](#) Security Considerations

The present memo does not introduce any new security issue that is not already mentioned in [[I-D.farinacci-lisp](#)] and [[I-D.bagnulo-lisp-threat](#)]. Nevertheless, we discuss hereafter some issues related to the reachability bits.

[10.1.](#) Reachability bits DoS

An attacker can deactivate a particular RLOC on a mapping of an ITR with a single packet using the reachability bits. If the reachability bit of a RLOC is set to one, the RLOC is reachable, otherwise it is unreachable.

Since reachability information on specific RLOCs can be modified by the reachability bits in the LISP header carried by data packets and not a control protocol, it is possible for an attacker to make a DoS on a EID by sending a single packet for that EID where all the reachability bits are set to zero. To succeed the attack, it is not required to have a bi-directional flow, the only constraint is to build a LISP packet, for an EID mapping present in the ITR, with as source EID the one that is meant to be made unreachable and a correctly formed LISP header having reachability bits all set to zero. Once the packet has been received, all RLOCs will be set to unreachable, and the ITR will not be able to reach the EID used as source, until another packet (not spoofed) will set again the RLOCs to a reachable state.

To tackle this issue two solutions are available:

- o Keep the reachability bits semantic, but add a confirmation phase to be sure the RLOC must be deactivated. When a reachability bit has changed compared to the mapping present in the cache, a Map-Request should be sent in order to obtain the new mapping with the correct reachability information. In OpenLISP, this can be easily implemented, since for any reachability change, a message is sent through the mapping sockets in order to inform the mapping distribution system, which in turn will perform the request.
- o Change the reachability bits semantic to become a version number. Instead of carrying reachability information for each RLOC, the bits contain the version number of the mapping. If the version number changes for an EID, a mapping request is sent.

The two solutions are not "bulletproof", however, they can help in removing, or at least reducing, DoS attacks. Nevertheless, a control on the rate of Map-Request is needed in order to avoid DoS attacks on the mapping system. The solution based on version number is more

DoS-proof as the attacker must be able to know the version number to launch the attack. If the version number is not near the real version number, the message can be considered as invalid.

11. Informative References

[FreeBSD] The FreeBSD Project, "FreeBSD, the power to serve",
<<http://www.freebsd.org>>.

[I-D.bagnulo-lisp-threat]
Bagnulo, M., "Preliminary LISP Threat Analysis",
[draft-bagnulo-lisp-threat-01](#) (work in progress),
July 2007.

[I-D.farinacci-lisp]
Farinacci, D., Fuller, V., Oran, D., and D. Meyer,
"Locator/ID Separation Protocol (LISP)",
[draft-farinacci-lisp-07](#) (work in progress), April 2008.

- [I-D.iab-raws-report]
Meyer, D., "Report from the IAB Workshop on Routing and Addressing", [draft-iab-raws-report-02](#) (work in progress), April 2007.
- [I-D.irtf-rrg-design-goals]
Li, T., "Design Goals for Scalable Internet Routing", [draft-irtf-rrg-design-goals-01](#) (work in progress), July 2007.
- [I-D.templin-seal]
Templin, F., "The Subnetwork Encapsulation and Adaptation Layer (SEAL)", [draft-templin-seal-22](#) (work in progress), June 2008.
- [I-D.van-beijnum-multi-mtu]
Beijnum, I., "Extensions for Multi-MTU Subnets", [draft-van-beijnum-multi-mtu-02](#) (work in progress), February 2008.
- [NetProg] Stevens, W., Fenner, B., and A. Rudoff, "UNIX Network Programming, The Sockets Networking API.", Addison-Wesley Professional Computing Series Volume 1 - Third Edition, 2004.
- [TCPIP] Wright, G. and W. Stevens, "TCP/IP Illustrated Volume 2, The Implementation.", Addison-Wesley Professional Computing Series, 1995.

[Appendix A](#). Man Pages

The following sections contain the manpages that can be obtained on a FreeBSD system once OpenLISP has been completely installed.

[A.1](#). `map(1)`

NAME

map -- manually manipulate the LISP mappings

SYNOPSIS

map [-dnqtv] command [[modifiers] args]

DESCRIPTION

The map utility is used to manually manipulate the network mapping tables (both cache and database). Only the super-user may modify the mapping tables. It normally is not needed, as a system mapping table management daemon, such as LISP-ALT, should tend to this task.

The map utility supports a limited number of general options, but a rich command language, enabling the user to specify any arbitrary request that could be delivered via the programmatic interface discussed in map(4).

The following options are available:

- d Run in debug-only mode, i.e., do not actually modify the routing table.
- n Bypass attempts to print host and network names symbolically when reporting actions. (The process of translating between symbolic names and numerical equivalents can be quite time consuming, and may require correct operation of the network; thus it may be expedient to forget this, especially when attempting to repair networking operations).
- v (verbose) Print additional details.
- q Suppress all output from the add, change, and delete commands.

The map utility provides five commands:

add Add a mapping.

delete	Delete a specific mapping.
get	Lookup and display the mapping for an EID.
monitor	Continuously report any changes to the mapping information base, mapping lookup misses, etc.
flush	Remove all mappings. This includes mappings from both the cache and the database.

The monitor command has the syntax:

```
map [-n] monitor
```

The flush command has the syntax:

```
map [-n] flush
```

The other commands have the following syntax:

```
map [-n] command [-local] [-inet | -inet6] EID
[-inet | -inet6] RLOC [Priority [Weight [Rechability]]]
```

where EID is the address of the EID-Prefix (it can be also a full address), -local indicates if the mapping should be treated as part of the local mapping database or as part of the cache. Default is cache. The keyword -inet and -inet6 are not optional, they must be used before any address (both EID and RLOC). These keywords indicate if the following address should be treated as an IPv4 or IPv6 address/prefix. RLOC is the address of the RLOC argument. Likewise the EID, it must be preceded by -inet or -inet6 keyword in order to indicate the address family. The EID must be specified in the net/bits format. For example, -inet 128.32 is interpreted as -inet 128.0.0.32; -inet 128.32.130 is interpreted as -inet 128.32.0.130; and -inet 192.168.64/20 is interpreted as the network prefix 192.168.64.0 with netmask 255.255.240.0.

The values Priority, Weight, and Reachability are optional to declare.

If not declared, the following default values are set:

Priority	255 (Not usable)
Weight	100
Reachability	0 (not reachable)

It is not mandatory to declare all of them, but when declaring one, all the previous must be also declared. This means that to declare a weight the priority must also be declared; and to set the reachability to 1 (reachable) both priority and weight must be

declared.

Mappings have associated flags that influence operation. These flags may be set (or sometimes cleared) by indicating the following corresponding modifiers:

-static	MAPF_STATIC	- manually added mapping (default)
-nostatic	~MAPF_STATIC	- pretend mapping added by kernel or daemon

All symbolic names specified for an EID or RLOC are looked up first as a host name using `gethostbyname(3)`. If this lookup fails, `getnetbyname(3)` is then used to interpret the name as that of a network.

The map utility uses a mapping socket and the message types `MAPM_ADD`, `MAPM_DELETE`, `MAPM_GET`, and `MAPM_CHANGE`. The flush command is performed using the `sysctl(3)` interface. As such, only the super-user may modify the mapping tables.

EXAMPLES

The command to add a mapping, in the LISP database, for EID 1.1.0.0/16, having RLOC 2.2.2.2 and Priority 1, Weight 100, and marked as Reachable, is:

```
map add -local -inet 1.1.0.0/16 -inet 2.2.2.2 1 100 1
```

The command to delete the same mapping is:

```
map delete -inet 1.1.0.0
```

To add in the cache a mapping having several RLOCs, the command is:

```
map add -inet 1.1.0.0/16 -inet 2.2.2.2 1 100 1 -inet 3.3.3.3  
2 100 1
```

```
-inet 4.4.4.4 3 100 -inet 5.5.5.5
```

The above command associate to the EID-Prefix 1.1.0.0/16 the following RLOCs and related Priority, Weight, and Reachability values:

RLOC	Priority	Weight	Reachability
2.2.2.2	1	100	Reachable
3.3.3.3	2	100	Reachable

4.4.4.4	3	100	Unreachable
5.5.5.5	255	100	Unreachable

Iannone, et al.

Expires January 17, 2009

[Page 38]

Internet-Draft

OpenLISP Implementation Report

July 2008

EXIT STATUS

The map utility exits 0 on success, and >0 if an error occurs.

SEE ALSO

netintro(4), map(4), mapstat(1),

L. Iannone and O. Bonaventure, OpenLISP Implementation Report,
[draft-iannone-openlisp-implementation-01.txt](#).

D. Farinacci, V. Fuller, D. Oran, and D. Meyer, Locator/ID
Separation protocol (LISP), [draft-farinacci-lisp-07.txt](#).

NOTE

Please send any bug report or code contribution to the authors
of OpenLISP.

AUTHORS

Luigi Iannone <luigi.iannone@uclouvain.be>

HISTORY

The map utility appeared in FreeBSD 7.0.

BSD

July 15, 2008

BSD

[A.2.](#) map(4)

MAP(4)

BSD Kernel Interfaces Manual

MAP(4)

NAME

map -- kernel LISP mapping cache and database

SYNOPSIS

```
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/maptable.h>
```

```
int
socket(PF_MAP, SOCK_RAW, int family);
```

DESCRIPTION

OpenLISP provides some mapping facilities into the kernel of FreeBSD. The kernel maintains a mapping information database, which is used in selecting the appropriate RLOCs when transmitting/forwarding packets.

A user process (or possibly multiple co-operating processes) maintains this database by sending messages over a special kind of socket. Mapping table changes may only be carried out by the super user.

The operating system may spontaneously emit mapping messages in response to external events, such as receipt of a packet for which no mapping is available. The message types are described

in greater detail below.

When handling a packet, the kernel will attempt to find the most specific EID mappings matching the source and the destination addresses. If there is more than one RLOC in the mapping the actual RLOC used for encapsulation is chosen following the rules described in (LISP). Note, however, that for local mappings, i.e., mappings that are part of the database, the flag "i" must be set in order for the RLOC to be used. This is to avoid to emit packets with a source address that does not belong to the machine. If no mapping is found, for both source and destination EIDs, packet is handed back to normal IP operation, which may lead to sending the packet, if the EID is routable and a route is available in the IP routing table.

One opens the channel for passing mapping control messages by

using the socket call. There can be more than one routing socket open per system.

Messages are formed by a header followed by a number of sockaddrs (of variable length depending on the address family) and RLOC metrics data structure.

Any messages sent to the kernel are returned, and copies are sent to all interested listeners. The kernel will provide the process ID for the sender, and the sender may use an additional sequence field to distinguish between outstanding messages. However, message replies may be lost when kernel buffers are exhausted.

The kernel may reject certain messages, and will indicate this by filling in the map_errno field. The OpenLISP code returns the following error codes if new mappings cannot be installed:

ENOBUFS: If insufficient resources were available to install a new mapping.

EEXIST: If the EID-Prefix already exist in the mapping table.

EINVAL: This error code can be returned in two cases. The first case is when the list of RLOC provided for a mapping

contains replicated addresses. The second case is when a "local" mapping is provided without any RLOC (address) belonging to the system.

A process may avoid the expense of reading replies to its own messages by issuing a `setsockopt(2)` call indicating that the `SO_USELOOPBACK` option at the `SOL_SOCKET` level is to be turned off. A process may ignore all messages from the mapping socket by doing a `shutdown(2)` system call for further input.

If a mapping is in use when it is deleted, the entry will be marked down and removed from the mapping table, but the resources associated with it will not be reclaimed until all references to it are released. User processes can obtain information about the mapping entry for a specific EID by using a `MAP_GET` message.

Messages include:

```
#define MAPM_ADD          0x1      /* Add Map */
#define MAPM_DELETE       0x2      /* Delete Map */
#define MAPM_CHANGE       0x3      /* Change Metrics or flags */
#define MAPM_GET          0x4      /* Report Metrics */
#define MAPM_MISS         0x5      /* Lookup Failed */
```

```
#define MAPM_BADREACH     0x6      /* Reachability Block Problem */
#define MAPM_REACH        0x7      /* Reachability Block Changed */
```

A message header consists of the following:

```
struct map_msghdr {
    u_short map_msglen;      /* to skip over non-understood
                             messages */
    u_char  map_version;     /* future binary compatibility */
    u_char  map_type;        /* message type */

    int     map_flags;       /* flags, incl. kern & message,
                             e.g. DONE */
    int     map_addrs;       /* bitmask identifying sockaddrs
                             in msg */
    int     map_rloc_count;  /* Number of rlocs appended to
                             the msg */
    pid_t   map_pid;        /* identify sender */
}
```

```

        int      map_seq;          /* for sender to identify action
                                   */
        int      map_errno;        /* why failed */

};

```

The ``int map_flags'' is as defined as:

```

#define MAPF_UP          0x1      /* mapping usable */
#define MAPF_LOCAL       0x2      /* Mapping is local
                                   * This means that it should be
                                   * considered
                                   * as part of the LISP Database
                                   */
#define MAPF_STATIC      0x4      /* manually added */
#define MAPF_DONE        0x8      /* message confirmed */

```

Specifiers for which addresses are present in the messages are:

```

#define MAPA_EID         0x1      /* EID sockaddr present */
#define MAPA_EIDMASK     0x2      /* netmask sockaddr present */
#define MAPA_RLOC        0x4      /* Locator present */

```

If MAPA_RLOC is set, it means that there are "int map_rloc_count" pairs of "struct sockaddr" and "struct rloc_mtx" present.

The ``struct rloc_mtx'' is as defined as:

```

struct rloc_mtx {
                                /* Metrics associated to the RLOC
                                   * Usefull for messages mapping

```

```

                                * sockets.
                                */
        u_int8_t priority;      /* Each RLOC has a priority.
                                   */
        u_int8_t weight;        /* Each locator has a weight.
                                   */
        u_int16_t flags;        /* Flags concerning specific RLOC.
                                   */
        u_int32_t mtu;          /* MTU for the specific RLOC.
                                   */
};

```


The ``u_int16_t flags'' is as defined as:

```
#define RLOCF_REACH    0x01 /* RLOC Reachable. */
#define RLOCF_LIF      0x02 /* RLOC is a local interface.
                             * This is only valid for local
                             * mappings.
                             */
```

A good example of how to use mapping sockets can be found in
/usr/src/sbin/map/map.c.

SEE Also

map(1), mapstat(1).

L. Iannone and O. Bonaventure, OpenLISP Implementation Report,
[draft-iannone-openlisp-implementation-00.txt](#).

D. Farinacci, V. Fuller, D. Oran, and D. Meyer, Locator/ID
Separation protocol (LISP), [draft-farinacci-lisp-07.txt](#).

NOTE

The MAPM_CHANGE message is not yet implemented.

Please send any bug report or code contribution to the authors of
OpenLISP.

AUTHORS

Luigi Iannone <luigi.iannone@uclouvain.be>

HISTORY

A PF_MAP protocol family has been introduced with OpenLISP on
FreeBSD 7.0.

BSD

July 15, 2008

BSD

Iannone, et al.

Expires January 17, 2009

[Page 43]

Internet-Draft

OpenLISP Implementation Report

July 2008

[A.3.](#) mapstat

MAPSTAT(1)

BSD General Commands Manual

MAPSTAT(1)

NAME

mapstat -- Modification of the netstat(1) utility to show LISP-related network status

DESCRIPTION

The mapstat command symbolically displays the contents of various network-related data structures. It is a modification of the existing netstat command, thus it basically offers the same identical features and can be used in the same identical way. Please refer to netstat(1) for more information on the normal use of netstat.

What mapstat introduces is that fact that it can show LISP-related network status. Where applies, mapstat accepts also the word "lisp" as protocol. Try the following command as an example:

```
mapstat -sf inet -p lisp
```

The mapstat adds the new following option:

-X Displays the content of mapping tables. The mapping table display indicates the available mappings and their status.

Each mapping consists of an EID, flags related to the whole mapping, references, and the list of RLOC. For each RLOC there is the address, its priority, its weight, and the flags related to that specific RLOC. It also shows the available MTU (Maximum Transmission Unit) for the specific RLOC, and the number of times that the RLOC has been selected for packet encapsulation.

The flags field shows a collection of information about the mapping or the RLOC stored as binary choices. The individual flags are the listed hereafter.

General flags

- U The mapping entry is "up" and usable.
- L The mapping entry "local", i.e., it is part of the lisp mapping database as defined in the original LISP proposal.
- S The mapping entry is "static", i.e., it has been

manually added.

RLOC specific flags

- R The specific RLOC is reachable.
- i The specific RLOC is a local interface. This flag can be set only for mappings that are part of the database, i.e., have the flag "L" set.

SEE ALSO

netstat(1), map(1), map(4)

NOTE

Please send any bug report or code contribution to the authors of OpenLISP.

AUTHORS

Luigi Iannone <luigi.iannone@uclouvain.be>

HISTORY

The mapstat utility appeared in FreeBSD 7.0.

BUGS

The code is still experimental. Some combinations of the -X option with other native options of netstat may not work or produce unexpected results.

Internet-Draft

OpenLISP Implementation Report

July 2008

Authors' Addresses

Luigi Iannone
UCLouvain, Belgium
Place St. Barbe 2
Louvain la Neuve, B-1348
Belgium

Email: luigi.iannone@uclouvain.be
URI: <http://inl.info.ucl.ac.be>

Damien Saucez
UCLouvain, Belgium
Place St. Barbe 2
Louvain la Neuve, B-1348
Belgium

Email: damien.saucez@uclouvain.be
URI: <http://inl.info.ucl.ac.be>

Olivier Bonaventure
UCLouvain, Belgium
Place St. Barbe 2
Louvain la Neuve, B-1348
Belgium

Email: Olivier.Bonaventure@uclouvain.be
URI: <http://inl.info.ucl.ac.be>

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

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.

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, THE IETF TRUST 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.

Intellectual Property

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.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).