The OpenLISP Control Plane Architecture

Dung Chi Phung, UPMC and VNU
Stefano Secci, UPMC
Damien Saucez, INRIA Sophia Antipolis
Luigi Iannone, Telecom ParisTech

Abstract

Among many options tackling the scalability issues of the current Internet routing architecture, the Locator/Identifier Separation Protocol (LISP) appears as a viable solution. LISP improves a network's scalability, flexibility, and traffic engineering, enabling mobility with limited overhead. As for any new technology, implementation and deployment are essential to gather and master the real benefits that it provides. In this article, we present the first complete open source implementation of the LISP control plane. Our implementation is deployed in the worldwide LISP Beta Network and the French LISP-Lab testbed, and includes the key standardized control plane features. Our control plane software is the companion of the existing OpenLISP data-plane implementation, allowing the deployment of a fully functional open source LISP network compatible with any implementation respecting the standards.

The Internet is suffering from scalability concerns, mainly due to the BGP routing infrastructure, and provides limited support to new advanced services. As discussed in [1, 2], a way to improve Internet scalability is separating the IP space into locator and identifier spaces. The Locator/Identifier Separation Protocol (LISP) [3] is henceforth being architected in this sense and introduces a two-level routing infrastructure on top of the current BGP+IP architecture, mapping an endpoint identifier (EID) to one or several routing locators (RLOCs). RLOCs remain globally routable, while EIDs become provider-independent and only routable in the local domain. The resulting hierarchical routing architecture opens the way to benefits ranging from BGP routing table size reduction and efficient traffic engineering, up to seamless IP mobility. Moreover, LISP enables a large set of applications and use cases such as virtual machine mobility management, layer 2 and layer 3 virtual private networks, intra-autonymous system (AS) traffic engineering, and stub AS traffic engineering.

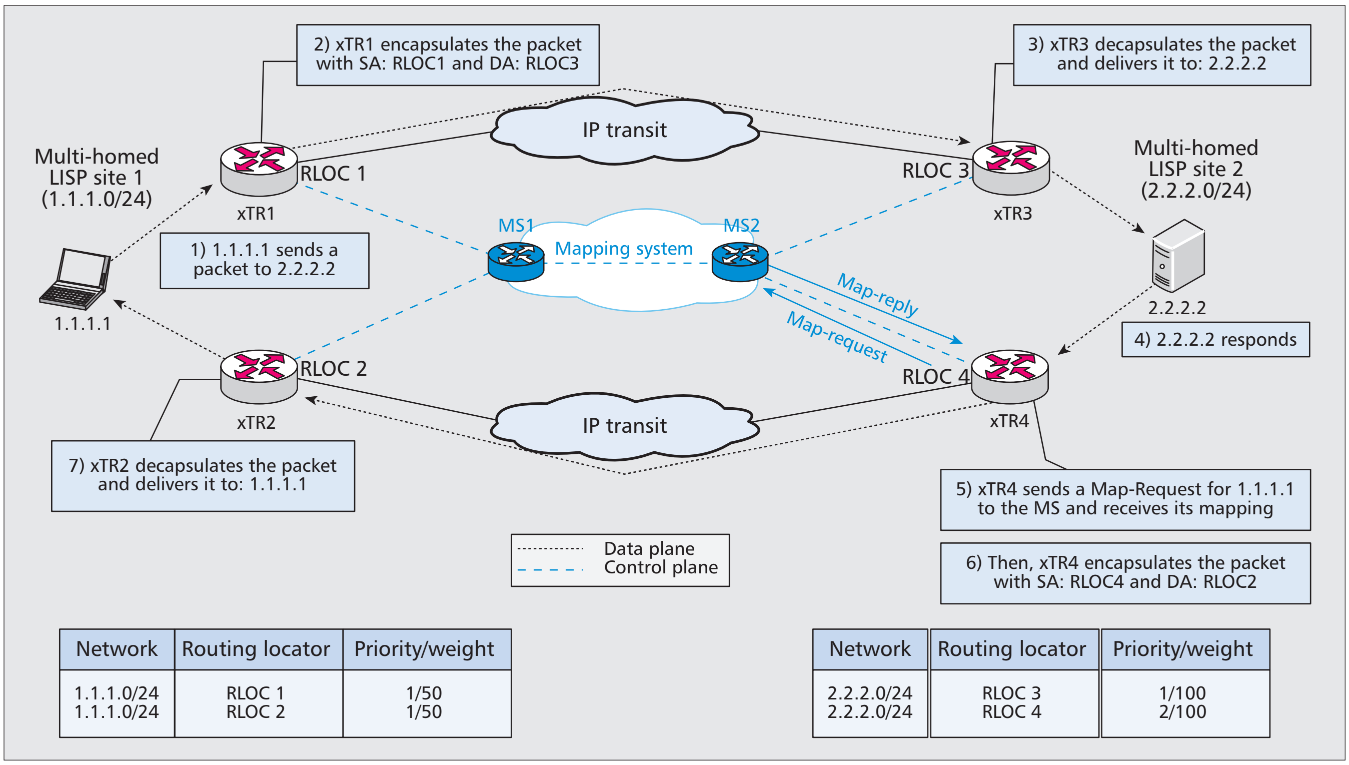
More technically, LISP uses a *map-and-encap* approach, where a mapping (i.e., a correspondence between an EID-Prefix and its RLOCs) is first retrieved and used to encapsulate the packet in a LISP-specific header that uses only RLOCs as addresses. Such a map-and-encap operation in LISP is performed using a distributed mapping database for the first packet of a new destination EID; then the mapping is cached locally for all subsequent packets. The LISP control plane is based on signaling protocols necessary to handle EID-to-RLOC registrations and resolutions, dynamically populating mapping caches at LISP network nodes. Since several RLOCs can be registered for the same EID, priority and weight metrics are associated with each RLOC in order to decide which one to use (highest priority) or how to do load-balancing (proportionally to the weights if priorities are equal) [4]. In practice, when a host sends a packet to another host at another LISP site, it sends a native IP packet with the EID of the targeted host as the destination IP address; the packet reaches a border router of the network that acts as an ingress tunnel router (ITR), maps EID to RLOCs, appends a LISP header and an external IP/UDP header with the ITR as source node, and, as the destination address, an RLOC selected from the mapping of the destination EID. The egress tunnel router (ETR) that owns the destination RLOC strips the outer header (i.e., decapsulates) and sends the native packet to the destination EID.

For example, in Fig. 1 the traffic from host 1.1.1.1 to host 2.2.2.2 is encapsulated by the ITR toward one of the RLOCs (the one with the highest priority, i.e., RLOC3), which acts as the ETR and decapsulates the packet before forwarding it to its final destination. On the way back to 1.1.1.1, RLOC4's xTR queries the mapping system and gets two RLOCs with equal priorities, hence performing load-balance as suggested by the weight metric.

The advantage of creating network control functions disjoint from the data plane is the possibility of programming the control plane independent of the forwarding logic, and thus to implement advanced and personalized functionalities, as done in [5] for instance, for virtual machine mobility management. This separation respects the software defined networking paradigm [6].

OpenLISP [7] is an open source implementation of the LISP data plane in a FreeBSD environment. As a standalone, an OpenLISP node is not able to handle all control plane signaling within a LISP network. Our control plane implementation aims at filling this gap, while keeping the data and control planes independent of each other for performance reasons, as detailed hereafter. Our control plane implementation is used to seamlessly interconnect the UPMC, Telecom ParisTech, INRIA, UNIROMA1, VNU, the University of Prague, and UFRJ LISP sites spread worldwide, and is deployed in the official LISP Beta Network.[1] We are also

[1] *LISP Beta Network worldwide testbed (website): http://www.lisp4.net*


2) xTR1 encapsulates the packet with SA: RLOC1 and DA: RLOC3
3) xTR3 decapsulates the packet and delivers it to: 2.2.2.2
IP transit
Multi-homed LISP site 1 (1.1.1.0/24)
RLOC 1
xTR1
RLOC 3
xTR3
Multi-homed LISP site 2 (2.2.2.0/24)
MS1
MS2
Mapping system
1) 1.1.1.1 sends a packet to 2.2.2.2
1.1.1.1
Map-reply
Map-request
2.2.2.2
4) 2.2.2.2 responds
RLOC 2
xTR2
RLOC 4
xTR4
IP transit
7) xTR2 decapsulates the packet and delivers it to: 1.1.1.1
5) xTR4 sends a Map-Request for 1.1.1.1 to the MS and receives its mapping
Data plane
Control plane
6) Then, xTR4 encapsulates the packet with SA: RLOC4 and DA: RLOC2

Network	Routing locator	Priority/weight
1.1.1.0/24	RLOC 1	1/50
1.1.1.0/24	RLOC 2	1/50

Network	Routing locator	Priority/weight
2.2.2.0/24	RLOC 3	1/100
2.2.2.0/24	RLOC 4	2/100

Figure 1. *An example of LISP communications between two LISP sites.*

using it in combination with OpenLISP as the standard implementation of nodes in the French LISP-Lab platform involving a dozen partners to date.[2] Our purpose is to boost LISP deployments by providing a full-fledged LISP open source software implementation, usable in operational networks and able to be freely tailored, to facilitate implementation of new ideas leveraging on LISP. Our integrated OpenLISP system is fully compatible with the standard as well as other implementations (e.g., Cisco IOS) as reviewed hereafter.

In the following, we detail the OpenLISP control plane architecture and implementation aspects before describing performance evaluation results.

The LISP Control Plane

For scalability reasons, ITRs learn mappings on-demand via the so-called *mapping system*. The mapping system is composed of the *mapping database system* and the *map-server interface* [8].

The mapping system workflow is summarized in Fig. 2. On one hand, the mapping database system constitutes the infrastructure that stores mappings on the global scale, potentially using complex distributed algorithms ([8–10]). On the other hand, the map-server interface hides this complexity via two network elements, the *map resolver* (MR) and *map server* (MS), deployed at the edge of the mapping database system, which LISP sites contact to retrieve and register mappings. More precisely, when an ITR is willing to obtain a mapping for a given EID, it sends a *Map-Request* message to an MR.

The MR is connected to the mapping database system and implements the lookup logic in order to determine at which LISP site the Map-Request must be delivered (to any of its ETRs), and delivers it. The ETR receiving the query will return the mapping directly to the requesting ITR with a *Map-Reply* message. It is worth noting that the ETR of a LISP site is not directly involved in the mapping database system but is instead connected to an MS. The ETR sends a *Map-Register* message to that MS, which later ensures that the mapping is registered in the mapping database system. Optionally, the MS can acknowledge the registration with a *Map-Notify* message.

Several mapping database systems have been proposed (e.g., [8–10]), but only the Delegated Database Tree (LISP-DDT, [10]) that we implement in our control plane is deployed. In LISP-DDT, the MR discovers where to send the Map-Request by iteratively sending Map-Requests and receiving *Map-Referral* messages via the hierarchical LISP-DDT infrastructure, similar to DNS [10].

The OpenLISP Control Plane Architecture

In this section, we describe the design of our OpenLISP control plane implementation, issued under a BSD licence.[3] Given that the main role of the LISP control plane is the management of EID-to-RLOC mappings with the mapping system, in the following we first focus on the design of the mapping database, and then we detail the different modules.

Mapping System and Key Network Nodes

The heart of the OpenLISP control plane is the EID-to-RLOC mapping database, synthetically referred to as *map-table* in the following. Each map-entry of the map-table consists of an EID prefix with a list of RLOCs, each RLOC associated with a structure that contains the RLOC address and related attributes (i.e., priority and weight). The three network elements involved in the control plane, ETR, MS, and MR, serve different purposes; hence, they implement their own map-table logic, as detailed hereafter.

ETR's map-entries correspond to the mappings for the dif-

[2] *ANR LISP-Lab project testbed website: http://www.lisp-lab.org.*

[3] *OpenLISP control-plane source code: https://github.com/lip6-lisp/control-plane*

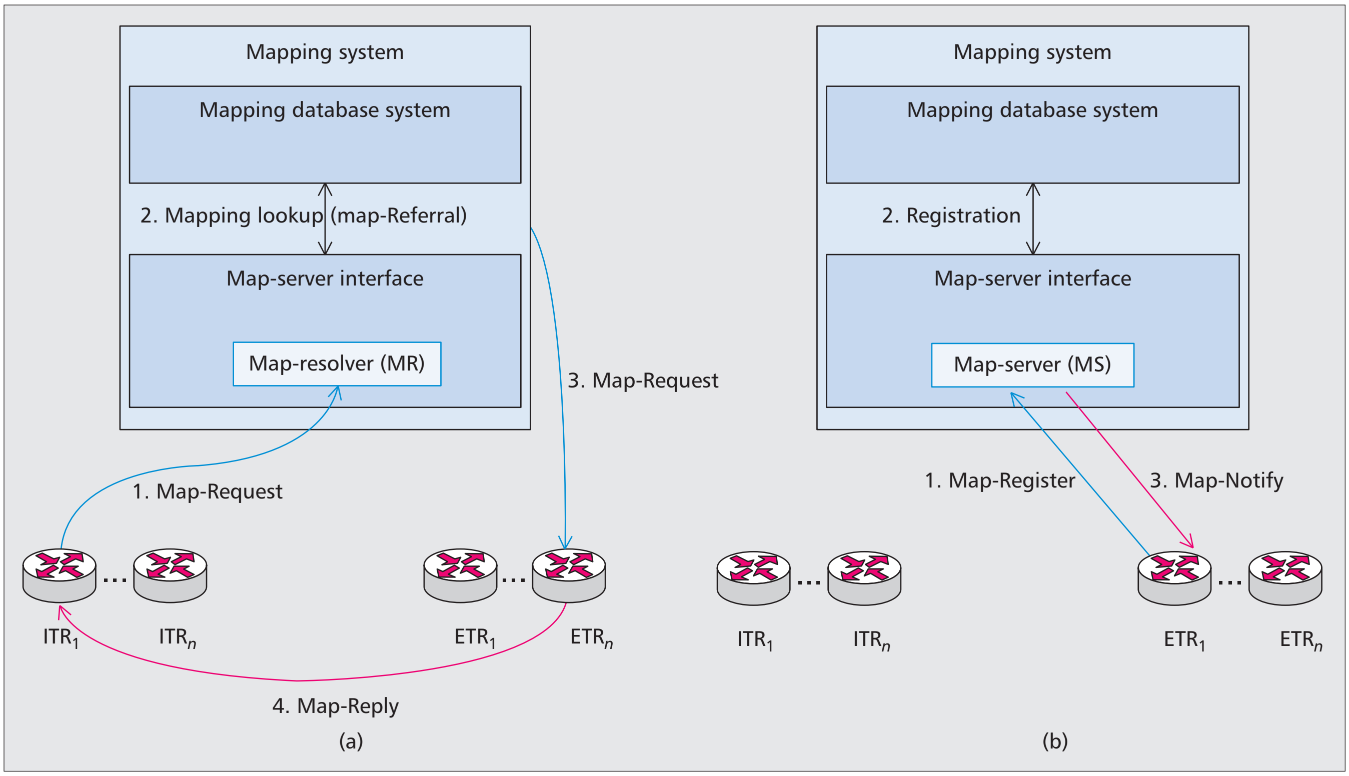


Figure 2. *LISP mapping system workflow: a) mapping retrieval; b) mapping registration.*

ferent EID prefixes of the LISP site it serves and should register via an MS. Each such map-entry must have at least one RLOC address.

Map-Servers maintain EID prefix registrations for the LISP sites they serve and for EID prefixes not assigned yet. Therefore, we distinguish the following two map-entry types:

- *Registered map-entries* are built on Map-Register messages received from ETRs and are associated with meta-information about the registering site (e.g., cryptographic keys authorized to register mappings, contact addresses). The MS can use these entries to directly reply to Map-Request messages on behalf of ETRs if commissioned to do so.
- *Negative map-entries* are used to define range of IP prefixes that belong to the EID space but do not require LISP encapsulation. Requests for such prefixes generate negative map-replies [8].

Map-Resolvers maintain a map-table to speed up mapping resolution, and we distinguish the next two types of entries:

- *Negative map-entries* are similar to an MS's negative map-entries. An MR hence immediately sends a negative Map-Reply for not yet assigned EID prefixes.
- *Referral map-entries* contain the addresses of other DDT nodes (MRs) that are supposed to provide more specific LISP-DDT mappings (i.e., have a longer EID prefix match).

Even though they are logically separated, map-tables are implemented within a compact radix tree data structure instance optimized for fast IP prefix lookup [11]. Actually, as our implementation is dual-stack, we maintain two radix tree instances, one for IPv4 EIDs and the other for IPv6 EIDs.

Control Plane Modules

Our control plane implementation includes the essential features to operate a multi-site LISP network, including all the LISP-DDT logic and complete support of both IPv4 and IPv6. In order to operate the control plane independent of the data plane, it is divided into independent modules with different functionalities (Fig. 3).

As depicted in Fig. 3, the control plane receives the messages from a dedicated queue, which gets them in turn from the kernel's UDP socket queue. The control plane is based on one general orchestration processes (i.e., *control*) and three specialized processes that implement MR, MS, and xTR network element logics. The treatment of mapping-resolution-related and registration-related messages within these processes is isolated thanks to the use of threads. Each process is composed of several modules, as described in the following.

The xTR process includes the following three modules:

MAP-REGISTER module: Implemented at the ETR interface; it sends periodic information (each 60 s, as recommended in [3]) about map-entry registration to at least one MS. Note that ETRs are authenticated by an MS using their preconfigured shared key.

In order to support mapping system multi-tenancy, going beyond the current standards, the module allows specifying different keys for different MSs to allow an xTR to join LISP networks managed by independent MS stakeholders.

MAP-REPLY module: Implemented at the ETR interface, it receives and processes Map-Requests coming from the ITR or MSs. According to the standard, Map-Requests must be encapsulated (Encapsulated Control Message, ECM) Map-Request when sent to MRs, but are sent natively to ETRs. Our implementation supports these two modes with any combination of IPv4/IPv6 encapsulation. Upon reception of a Map-Request, an ETR replies with the corresponding Map-Reply.

PLANE-INTERWORKING module: This module allows the control plane to interact with the data plane and hence to form a full-fledged OpenLISP xTR. In order to perform data plane functions, the OpenLISP data plane maintains a mapping information base (MIB) consisting of the LISP cache (storing short lived mappings in an on-demand fashion) and LISP database. OpenLISP also provides a low-level abstraction called *Mapping Socket* to add or remove mappings from the MIB locally on the machine (e.g., by means of a daemon or using the command line). This interworking module uses the control plane to maintain the database interacting with the data plane through the Mapping Socket [7].

The MS process includes the following two modules:

MAP-REGISTER module: Implemented at the MS interface, it receives Map-Register messages from ETRs and updates the MS map-table accordingly. The MS verifies the authenticity of the Map-Register messages and ensures that their EID-prefixes belong to the LISP sites of which it is in charge.

In normal operations, mappings of given sites are stable with time. However, the specification requires periodically re-registering mappings. Therefore, to improve performance, our control plane hashes the Map-Register message to check whether the mapping has changed since the last registration, complete registration being done only upon a mapping update. If the ETR asks for a notification, a Map-Notify message is sent back to the ETR.

MAP-REQUEST module: Upon Map-Request reception, the module has a choice between two actions, depending on the map-table entry that corresponds to the EID in the Map-Request. If the EID corresponds to the EID prefix of a registered map-entry, the MS sends a Map-Reply back or forwards the Map-Request to one of the RLOCs in the map-entry, depending on the value of the proxy bit in the Map-Register message. If, instead, the EID corresponds to a site managed by the MS but has no active registration, a negative Map-Reply is sent back.

The Map-Resolver process contains the following two modules:

MAP-REQUEST module: It accepts and processes Map-Requests from xTRs. For DDT signaling, the Map-Request follows the map-referral chain until it reaches an MS or an ETR, or the number of referral nodes it passed through exceeds the maximum allowed number. To speed up performance, the MR caches map-referral messages in its map-table so that it can reuse it for further Map-Requests covered by the EID prefix.

MAP-REFERRAL module: It accepts the LISP-DDT Map-Requests to which it replies with a map-referral message. We provide in the control plane package a sample configuration that can be used to set up a DDT root [10].

Finally, the **control process** aims to orchestrate other processes. It is in charge of receiving control plane messages from the LISP network and dispatching them to the appropriate control plane process. A first-in first-out (FIFO) queue is used to absorb demand burstiness and catch messages coming from the UDP socket kernel queue. This process also populates the map-table, used by control plane processes, according to the device configuration file.

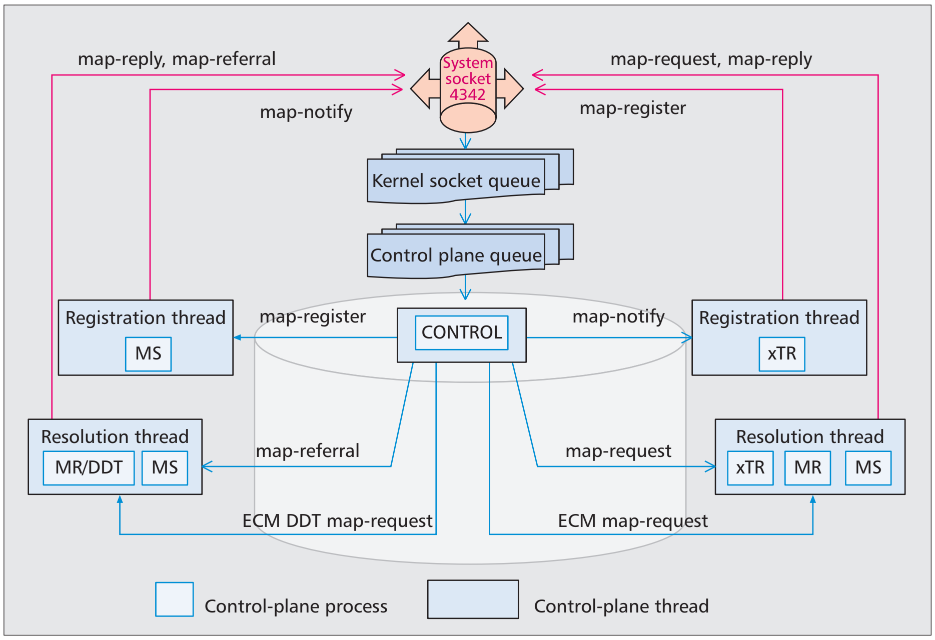


Figure 3. *System-level OpenLISP control plane multi-thread architecture.*

Running the OpenLISP Control Plane

The OpenLISP control plane process listens on the UDP 4342 LISP control port. It runs in the user space to allow easier programmability of its features, while the OpenLISP data plane runs in the kernel to give higher performance to data plane functions. Even though our control plane is designed for a FreeBSD environment, it can be adapted to Linux.

As depicted in Fig. 3, the control plane program handles three types of resident threads besides the main thread: one thread runs the control process, one thread is dedicated to mapping registrations, and the other threads are dedicated to Map-Request/Referral processing (resolution threads). The main thread accepts LISP control plane packets coming from the kernel socket queue and pushes them to a FIFO control plane queue in the user space based on a dynamic list. For load balancing, the control thread can dynamically create several resolution threads up to a maximum number, which is also left as a tunable parameter for the user via configuration files. The choice of using several pre-instantiated threads to process control plane messages and create a packet queue for the control plane fed by the kernel socket queue is dictated by scalability and robustness against attacks. It is worth noting that using multiple cores could create moderate processing time variances due to the dynamic thread-core binding operating system (OS) operations.

Finally, it is worth mentioning that a command line interface is also provided to allow an operator to interact with the control plane. More details on the configuration are provided in the documentation files of the software release.

Evaluation

We evaluated the performance of our LISP control plane by stressing an OpenLISP node running on a physical machine with a 2.67 GHz dual-core CPU and 2 Gbytes RAM. The evaluation focuses on the OpenLISP node system performance itself, independent of the LISP Beta Network topology. We do not account for packets not handled by the control plane due to drops in the network. Indeed, since LISP uses UDP to deliver both data plane and control plane messages, some of them may be dropped and definitely lost by intermediate nodes in an operational context, and the sender will eventually retransmit the packet after timeout. Therefore, the number of messages the control plane can handle depends on the provisioning of the kernel's UDP queue size, but also on the frequency with which the control plane program picks up packets from a kernel's queue and how fast it processes the messages. In order to avoid modifying the system configuration, we added in our control plane, more specifically in the control thread, a FIFO queue that is overprovisioned so that

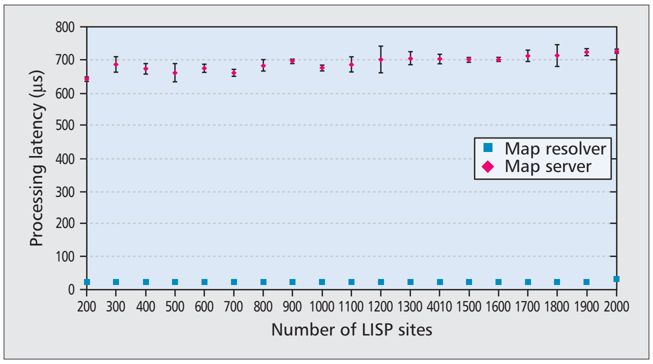


Figure 4. *Control plane processing latency as a function of the number of LISP sites.*

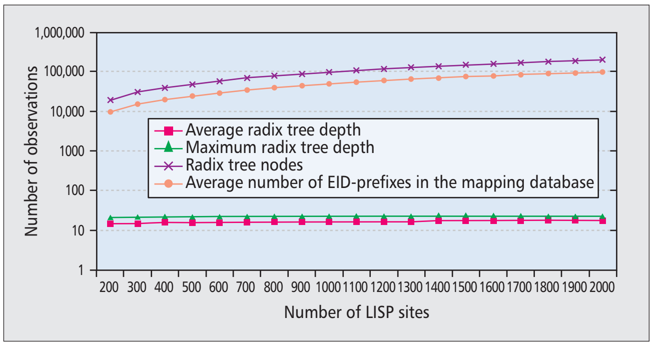


Figure 5. *Insight on the mapping database radix tree structure.*

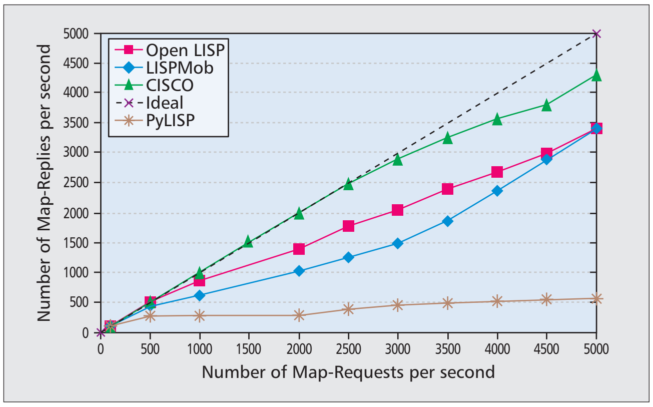


Figure 6. *Average number of received Map-Replies as a function of sent Map-Requests.*

the kernel's queue occupancy remains as small as possible. In the tests we used a control plane queue size of 100,000 packets; we tested the feasibility using smaller sizes (1000, 500, and 100), with no visible effects on performance, as well as with very high rates (more than 4000 packets/s).

In the following we evaluate the control plane processing latency. For the MS, it corresponds to the time taken to check the validity of the Map-Register message, update the mapping into the mapping system, and send back Map-Notify messages when needed. When generating the Map-Register messages in the tests, around 5 percent are randomly set to require a Map-Notify. For the MR, the processing latency corresponds to the mapping lookup time and the time to send the Map-Reply back. Figure 4 displays the processing latency for both MS and MR as a function of the number of connected LISP sites (i.e., the number of different mappings).

To carry out our measurements, we use a LISP site composed of two xTRs and one MS/MR node. xTRs send traffic at the maximum rate over a 100 Mb/s link with the MS/MR, stressing the control plane with almost 3000 packets/s in the control plane input queue. For the sake of realism, we fed the EID-to-RLOC database IPv4 prefixes of the DANTE public routing table,[4] fragmenting /16 prefixes into /24 prefixes: thus, we obtain a mapping database of more than 260,000 different EID prefixes. Randomly picking up EID prefixes from the database, we construct *s* sites, each site having from 1 to *e* EID prefixes (e.g., for multihoming TE or IP mobility management). We vary *s* from 200 to 2000 (roughly from one to 10 times the number of sites currently connected to the LISP Beta Network), with a step of 100 sites; *e* takes a random value between 1 and 100, so as to also include LISP sites intensively performing multihoming traffic engineering and IP/prefix mobility across RLOCs. It is worth noting that the number of RLOCs does not influence radix tree size. Once this setting is loaded in the MS/MR node, one of the xTR is used to send map-register messages for all sites to the MS, while the other xTR to send Map-Request messages to the MR. To prevent time-dependent artifacts, control plane messages are sent sequentially in about 20 different spaced test periods, with 20 messages sent per period on average. To avoid biases, the two signaling flows have not been sent concurrently.

Figure 4, showing both average and 99 percent confidence intervals of the obtained results, leads to two main conclusions. First, the processing time increases only by a limited amount, roughly 10 percent, while increasing the number of registered LISP sites from 200 to 2000, for both MS and MR. This result suggests that the logic implemented for the lookup represents a light portion of the overall processing load. We verified and the processing latency slightly decreases at certain steps with respect to the previous step because the second core started being used by the operating system. The relatively remarkable variance is likely the symptom of CPU differently assigning threads to cores at different executions.

Furthermore, under such severe load conditions, the Map-Server processing latency stays at very acceptable values (around 700 µs) for the provided computing power, and is about 30 times higher than the Map-Resolver latency; this is essentially due to the very high number of sites and prefixes to register, the fact that first, Map-Register messages need to be authenticated via HMAC, and then the mapping database possibly may need to be updated (hence roughly a quadratic time complexity). Map-Reply and Map-Notify messages that are close in size and written with a linear complexity have a similar light impact on the processing latency of MR and MS, respectively.

The processing latency performance also depends on the dispersion of the EID-prefix in the mapping database, which, as already mentioned, is built using a radix tree [11]. Figure 5 reports the average radix tree depth, the maximum tree depth, the total number of nodes, and the average number of EID prefixes in the mapping database (obviously, the same for the MR and MS cases; the confidence intervals are not visible). It is worth noting that the number of tree nodes is slightly higher than the total number of EID prefixes because of the necessary addition of branching nodes in the radix tree. Figure 5 shows that when the number of registered LISP sites increases, the radix tree depth does not increase significantly, despite the fact that the total number of nodes (directly affecting the size of memory used to store and manage the mapping database) increases exponentially. This explains why the number of LISP sites, as shown in Fig. 4, only marginally affects the processing latency.

[4] *DANTE project (http://www.dante.net)*

Our evaluation shows that our control plane implementation is scalable and offers the level of performance needed for operational deployment suffering from very high loads. Moreover, the overhead due to LISP encapsulation is proven to be negligible with the OpenLISP data plane implementation [7]. These results and our efforts to be in conformance with the standards position the combination of the OpenLISP data plane and our control plane implementation as a viable and efficient alternative to closed-source vendor-specific implementations. The proof is that one DDT root using our implementation is already integrated in the official LISP Beta Network control plane.[5]

Features	OpenLISP	Cisco IOS	LispMob	PyLISP	FritzBox
xTR	Yes	Yes	Yes	Yes	Yes
Map-Server	Yes	Yes	No	No	Yes
Map-Resolver	Yes	Yes	No	No	Yes
DDT	Yes	Yes	No	No	No
Open source	Yes	No	Yes	Yes	No
In-kernel data plane	Yes	Yes (monolithic)	No	No	N/A
LISP-TE	Yes	No	No	No	N/A
Proxy xTR	Yes	Yes	No	No	Yes

Table 1. *Comparison between LISP implementations (as of January 6, 2014).*

Related Work

Among the existing LISP implementations (OpenLISP, Cisco IOS,[6] FritzBox,[7] LISPMob,[8] PyLISP[9]), three are open source: OpenLISP, LISPMob, and PyLISP. The former is already described above since our control plane is built on the OpenLISP data plane. LISPMob is a multi-platform implementation of the LISP mobile node (LISP-MN) variant [12] intended for mobile devices (e.g., smartphones); in LISP-MN, mobile nodes are full-fledged xTRs relying on a lightweight version of the control plane. LISPMob is implemented in the user space and compatible with Linux and Android. Even though LISPMob is intended for mobile devices, it does not preclude its usage on routers; however, the limited control plane functionalities to date and its user space implementation would make it innapropriate for large-scale operational networks. PyLISP, a recent Python implementation of LISP, only provides xTR functions, and is also a pure user space implementation. The design of the other proprietary implementations (i.e., Cisco IOS[6] and FritzBox[7]) is unfortunately not well documented.

Table 1 compares the LISP implementations (for the FritzBox there is no public complete information to date); all respect the reference RFC [3] and are hence interoperable. The OpenLISP and Cisco IOS implementations are the most complete. Morever, to the best of our knowledge, OpenLISP is the only one supporting LISP traffic engineering (LISP-TE) [13] and map versioning, as well as the only open source implementation supporting Proxy-ITR/ETR features.

We quantitatively compared these implementations by measuring their reliability when replying to Map-Request messages.[10] Figure 6 gives the scatter plot of the Map-Request rate vs. the Map-Reply rate for an increasing Map-Request rate. Ideally, the Map-Reply rate should be equal to the Map-Request rate, but because of processing time and buffer limitations, some requests are eventually dropped. OpenLISP, LISPMob, and PyLISP were executed in the same single-core node of 2.67 GHz and 1 GB of RAM. We ran the Cisco implementation of a multi-core carrier grade router, the 3900 one, since tested lower-grade Cisco routers did stop the LISP control plane when approaching a few thousand Map-Requests per second. Results between the open source implementation and the Cisco implementations are therefore not directly comparable, but are reported for the sake of clarity. The Cisco one consequently appears as the most robust implementation, dropping about 10 percent of the control plane messages, only starting at around 4000 messages/s. Among the open source implementations, OpenLISP slightly outperforms LISPMob for low and mid-range rates, despite the additional features to manage, but has similar performance at higher rates. PyLISP in its current implementation is not very scalable and shows very poor performance already at 500 Map-Requests/s. Overall, these results show that the more mature implementations are those with a longer history.

Perspectives

Thanks to our development effort, OpenLISP is today the de facto single fully featured open source LISP implementation available. We hope that this will help boost the research in the field and deployment, and to improve the understanding and the insight of such a new technology as LISP.

Our performance evaluation combined with the data plane performance evaluation in [5] shows that our implementation is scalable enough for large networks and reaches performances suitable for real deployments. Our implementation is currently mature enough to be deployed in operational networks, and is actually used to interconnect at least seven LISP sites to the worldwide LISP Beta Network testbed and 11 to the LISP-Lab testbeds, correctly handling both data plane and control plane operations. Moreover, we have just integrated an OpenLISP DDT root server into the current worldwide LISP DDT hierarchy.[11]

We are currently enhancing the traffic engineering features to support various working modes concurrently, and we plan to add security features, integrating the related upcoming Internet Engineering Task Force (IETF) specification on the matter. We recently ported the control plane to the Linux environment; another important milestone already planned is to port the data plane to Linux as well, and the whole OpenLISP node to other BSD flavors (e.g., OpenBSD and NetBSD).

[5] *See http://ddt-root.org*

[6] *LISP Cisco IOS implementation (website): http://lisp.cisco.com*

[7] *FritzBox (website): http://www.fritzbox.eu/en/index.php*

[8] *LISPMob open source project (website): http://www.lispmob.org*

[9] *PyLISP open source (website): https://github.com/steffann/pylisp*

[10] *FritzBox has not been tested because hardware was not available to us.*

[11] *99 percent error bars are not included because they are so small that the results are not visible.*

Acknowledgments

This article was partially supported by the NU@GE project (http://www.nuage-france.fr), funded by the French Investissement d'avenir research programme, and the ANR LISP-Lab Project (http://www.lisp-lab.org; Grant No: ANR-13-INFR-0009) and the EIT ICT-Labs Future Networking Solutions action line.

References

[1] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB Workshop on Routing and Addressing," IETF RFC 4984, 2007.

[2] T. Li, Ed., "Recommendation for a Routing Architecture," IETF RFC 6115, 2011.

[3] D. Farinacci *et al.*, "Locator/ID Separation Protocol (LISP)," IETF RFC 6830, Feb. 2013.

[4] S. Secci, K. Liu, and B. Jabbari, "Efficient Inter-Domain Traffic Engineering with Transit-Edge Hierarchical Routing," *Computer Networks*, vol. 57, no. 4, Mar. 2013, pp. 976–89.

[5] P. Raad *et al.*, "Achieving Sub-Second Downtimes in Large-Scale Virtual Machine Migrations with LISP," *IEEE Trans. Network and Service Management*, in press.

[6] "Software Defined Networking: The New Norm for Networks," white paper, ONF, Apr. 2012.

[7] L. Iannone, D. Saucez, and O. Bonaventure, "Implementing the Locator/ID Separation Protocol: Design and Experience," *Computer Networks*, vol. 55, no. 4, Mar. 2011, pp. 948–58.

[8] V. Fuller and D. Farinacci, "LISP Map-Server Interface," IETF RFC 6833, Feb. 2013.

[9] V. Fuller *et al.*, "LISP Alternative Topology (LISP+ALT)," IETF RFC 6835, Feb. 2013.

[10] V. Fuller *et al.*, "LISP Delegated Database Tree," draft-fuller-lisp-ddt-04, Sept. 2012.

[11] G. Wright and W. Stevens, *TCP/IP Illustrated Volume 2, The Implementation*, Professional Computing Series, Addison-Wesley, 1995.

[12] D. Farinacci *et al.*," LISP Mobile Node," draft-meyerlisp-mn-09, July 2013.

[13] D. Farinacci, P. Lahiri, and M. Kowal, "LISP Traffic Engineering Use-Cases," draft-farinacci-lisp-te-04, Jan 2014.

Biographies

DUNG CHI PHUNG (Chi-Dung.Phung@upmc.fr.) received an M.Sc. degree from Vietnam National University (VNU), Hanoi, where he worked as a campus network engineer. He is on a leave of absence from VNU and working as a research engineer at Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, France.

STEFANO SECCI [M] (Stefano.Secci@upmc.fr) is an associate professor at Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, France. He received a dual Ph.D. degree from the Politecnico di Milano and Telecom ParisTech. He has also covered positions at NTNU, George Mason University, Fastweb Italia, and Ecole Polytechnique de Montréal. His current research interests are Internet resiliency and cloud networking. He is Vice-Chair of the Internet Technical Committee, joint between the IEEE Communications Society and the IEEE Internet Society (ISON).

DAMIEN SAUCEZ (Damien.Saucez@inria.fr) is a postdoctoral researcher working on information-centric networking (ICN) and software defined networking at INRIA Sophia Antipolis, France. His research interests include future Internet architecture and, in particular, traffic engineering and large-scale Internet measurements. He actively contributes to IETF standardization and implementation efforts. He has a Ph.D. in applied sciences from Université catholique de Louvain, Belgium.

LUIGI IANNONE (Luigi.Iannone@telecom-paristech.fr) is an associate professor at Telecom ParisTech. His research interests include intra- and inter-domain routing, future Internet architectures, mobility, wireless networks, and wired/wireless convergence. He has a Ph.D. in computer science from Université Pierre et Marie Curie (UPMC — Paris VI). He is Secretary of the IETF LISP working group.