ACCEPTED FROM OPEN CALL

LISP: A Southbound SDN Protocol?

Alberto Rodriguez-Natal, Marc Portoles-Comeras, Vina Ermagan, Darrel Lewis, Dino Farinacci, Fabio Maino, Albert Cabellos-Aparicio

ABSTRACT

The Locator/ID Separation Protocol (LISP) splits current IP addresses overlapping semantics of identity and location into two separate namespaces. Since its inception the protocol has gained considerable attention from both industry and academia, motivating several new use cases to be proposed. Despite its inherent control-data decoupling and the abstraction and flexibility it introduces into the network, little has been said about the role of LISP on the SDN paradigm. In this article we try to fill that gap and analyze if LISP can be used for SDN. The article presents a systematic analysis of the relevant SDN requirements and how such requirements can be fulfilled by the LISP architecture and components. This results in a set of benefits (e.g. incremental deployment, scalability, flexibility, interoperability, and inter-domain support) and drawbacks (e.g. extra headers and some initial delay) of using LISP for SDN. In order to validate the analysis, we have built and tested a prototype using the LISPmob open-source implementation.

INTRODUCTION

The Locator/ID Separation Protocol (LISP) [1] decouples identity from location on current IP addresses by creating two separate namespaces: endpoint identifiers to identify hosts, and routing locators to route packets. The original purpose of LISP was to solve the scalability issues of the Internet default-free zone (DFZ) routing tables by pushing traffic engineering practices to the identifiers space while keeping the locators space quasi-static and highly aggregatable. At the time of this writing LISP has been deployed in a pilot network (lisp4.net) that includes more than 20 countries and hundreds of institutions. LISP hardware and software are also widely available, both in open-source (lispmob.org, openlisp.org) and proprietary implementations (lisp.cisco.com).

Since its inception, LISP has gained significant traction in both industry and academia. As a result of LISP standardization and research efforts, the protocol has grown architecturally and has been applied to use cases beyond its original purpose. There is a growing interest in the role of LISP in Software Defined Networking (SDN) [2]. LISP is already becoming part of SDN solutions, such as the OpenDaylight controller (opendaylight.org). In this article we analyze the relation between the LISP architecture and the SDN paradigm.

There are two well-defined parts in any SDN deployment: the northbound and the southbound interfaces. The northbound offers a high-level application programming interface, where control applications can be deployed. The southbound is a low-level interface used to operate with the raw network elements. Currently, there is ongoing effort to define the high level abstraction interface (see Frenetic [3] or Procera [4] as examples). There are also several options with respect to the southbound interface, with OpenFlow [5] attracting the most interest from industry.

The main contribution of this article is to analyze LISP as a southbound SDN protocol. For this, the article presents a systematic analysis of the fundamental SDN requirements, inferred from the literature [2–10], and how such requirements can be fulfilled by the LISP architecture and components. The analysis results in a set of qualitative advantages and drawbacks as well as recommended potential improvements to overcome the identified issues. In order to validate the analysis, we build and test a prototype using the LISPmob open-source implementation (lispmob.org).

BACKGROUND: LISP OVERVIEW

The Locator/ID Separation Protocol (LISP) decouples host identity from its location. It creates two different namespaces: endpoint identifiers (EIDs) and routing locators (RLOCs). Hosts are identified by an EID, and their point of attachment to the network by an RLOC. To keep LISP incrementally deployable, in its very basic form EIDs and RLOCs are syntactically identical to current IPv4 and IPv6 addresses. However, the protocol allows arbitrary address families (e.g. MAC) to be used.

Figure 1 depicts the LISP common operation. Packets are routed based on EIDs within host sites and on RLOCs on transit networks. Since host A and host B are in different sites (e.g. two offices geographically separated), the packets from A to B have to traverse a transit network

Alberto Rodriguez-Natal and Albert Cabellos-Aparicio are with Technical University of Catalonia.

Marc Portoles-Comeras, Vina Ermagan, Darrel Lewis, and Fabio Maino are with Cisco Systems, San Jose, CA.

Dino Farinacci is with lispers.net.

0163-6804/15/$25.00 © 2015 IEEE

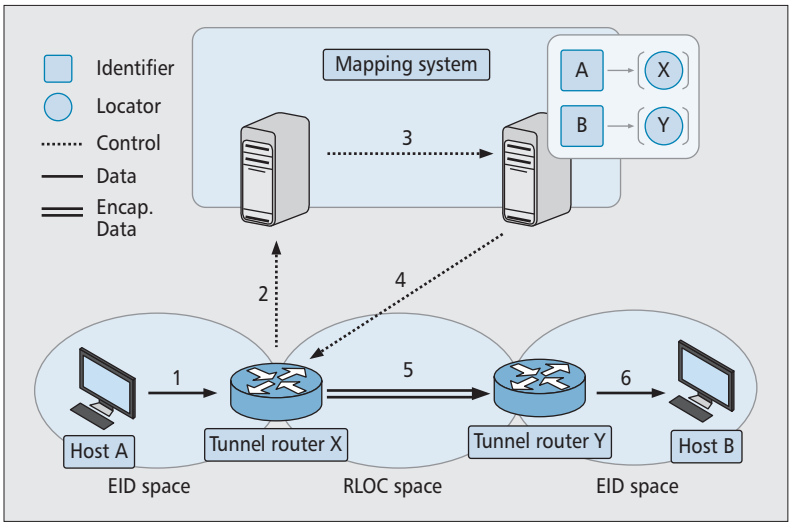

Figure 1. LISP overview.

(e.g. the Internet). To allow transit between the EID and RLOC spaces, LISP follows a map-and-encap approach performed by LISP tunnel routers deployed at edge points. In the image, tunnel router X receives the packet from host A addressed to host B (1). It knows that host B is in a different EID site, but it does not know where to reach that site (i.e. its RLOC). Tunnel router X requests this information to the mapping system (2). The LISP mapping system is a distributed database that stores EID to RLOC mappings. Tunnel router Y has previously registered its location and the set of EIDs it is in charge of in one of the mapping system internal servers. The mapping system routes the request internally (3) to find that server, and eventually it replies back with the requested location (4). Tunnel router X gets this information and caches it for future use. From now on, all EID packets from host A to host B will be encapsulated into an RLOC packet in tunnel router X and routed toward tunnel router Y (5). Upon arrival at the destination, tunnel router Y will decapsulate the packets and forward them natively to host B (6).

LISP: An SDN Architecture?

In this section we analyze if the LISP architecture, in its current form, can fulfill the requirements stemming from the SDN paradigm. Even though a formal definition of such requirements cannot be found in the literature, we infer the key SDN requirements by revisiting the design principles of the state-of-the-art SDN literature.

Control-Data Decoupling: One of the main reasons that motivated the emergence of OpenFlow [5] was to decouple the network control from the data forwarding devices. With its mapping system in place, LISP is capable of maintaining a distributed database where the network state and control information are stored. This database can be updated and queried by the LISP network elements in real time, and any change on it is propagated over the network. With this approach LISP is effectively decoupling control from data: while the data-plane remains at the router level, implemented on the tunnel routers, all control is pushed to the mapping system.

Network Programmability: Frenetic [3] and Procera [4] are two examples of the interest of the community in programing the network and improving its management. The LISP paradigm does not program the network but rather the mapping system. The control policies can be programmed and stored on the mapping system, then the LISP data-plane will operate accordingly. LISP semantics are poor when compared to state-of-the-art languages [3, 4] and focuses on representing the network state, therefore LISP should be complemented by a rich northbound language.

Centralized Control: Levin *et al.* [7] demonstrate that one core benefit of SDN is that it enables the network control logic to be designed and operated on a global network view, as though it were a centralized application. Since the LISP mapping system stores all network control state data and can be remotely accessed and updated in real time, it provides a global view of the network that effectively centralizes the control.

Scalability: Yeganeh *et al.* [8] show the concern of the SDN community about SDN scalability. LISP is a pull-based architecture that stores the network state information in the mapping system, and network entities (e.g. LISP tunnel routers) retrieve and cache only locally relevant state information on demand. Furthermore, the literature shows that the mapping system internals can be designed to be scalable [11].

Core-Edge Split: Casado *et al.* [9] analyze the main shortcomings of existing SDN architectures and point to the Fabric architecture as a solution. Fabric is based on an element called network fabric, a set of forwarding elements whose main function is packet forwarding. By taking base on this concept, they split the network into three components: hosts, edge switches, and core fabric. With this, rich network services such as isolation, mobility, or security are performed at the edge while fabric control is only responsible for packet forwarding. It is simple to establish a bijective relationship from Fabric components to LISP elements: tunnel routers perform edge switch functions, hosts are located on the EID space, and the core fabric corresponds to the RLOC space. From an abstract point of view, LISP offers an equivalent architecture to the one proposed by Fabric.

LISP SDN Building Blocks

In this section we analyze how specific LISP architectural elements can be used as SDN building blocks to understand the technical advantages and disadvantages of LISP as an SDN solution.

Flexible Namespace

The main LISP specification assumes IPv4 and IPv6 as address families, but it is flexible enough to allow using any other address families (e.g. MAC addresses). The LISP canonical address format (LCAF) allows defining ad-hoc address types that can be used for any purpose on a LISP system.

The template to define this type of address follows a simple TLV format (type-length-value). With this format, it is possible to define any

address type, including nested addresses of the same or different type. There are several address types defined at the time of this writing: AS number, geo-coordinates, application data, NAT-traversal data, multicast info, and so on. As an example, geo-coordinates addresses are used to carry geographical information along with any other address.

In general, such addresses allow LISP to map from any kind of identifier to any kind of locator, which means that, from an abstract point of view, LISP can map from any namespace to another. This address agnosticism enables rich network state programmability and can help to ease the interoperability challenges of heterogeneous SDN deployments

Distributed Mapping Database

Interface: The interface to exchange information with the mapping system is standard and open, and all the mapping system internal elements are hidden behind this interface (see Fig. 2). This allows the LISP data-plane devices to remain agnostic of the mapping system internal implementation. Such decoupling was put into test when the LISP beta-network deployed on the Internet (lisp4.net) replaced the existing mapping system, based on BGP, to a new one based on DNS without interfering with any of the LISP data-plane elements.

Arbitrary Information: Using the LISP flexible addresses (LCAF) described in the previous section, the mappings can contain any arbitrary information and be read/written from the mapping system using a standard interface. An SDN system can take advantage of this feature to store the network state. This is similar to what Onix [10] does with its own distributed databases.

Onix is a well known wide-area SDN deployment that addresses the lack of a general SDN control platform that can provide network-wide management abstractions. Onix provides an infrastructure to manage network state, on top of which different control-plane applications can be implemented. To offer this, Onix deploys its own database system to keep the network state and relies on the OpenFlow protocol to communicate with the network devices. Onix takes care of keeping consistent and distributed this network state over all network elements. With a LISP deployment, Onix could take advantage of LISP capabilities to provide similar functionality. First, it could use the mapping system with flexible addresses to keep network state and policies instead of deploying its own database system. Second, it could automatically reflect this state on the actual network if the network devices directly pull these policies from the mapping system using the LISP protocol.

Internal Scalability: The internal architecture of a specific mapping system varies depending on the type of information it is expected to store. Figure 2 also shows how the mapping system can use different internal implementations.

A mapping system indexing common IP addresses benefits from a hierarchical structure, such as DNS. This is the approach followed by the delegated database tree (DDT) based on [11],

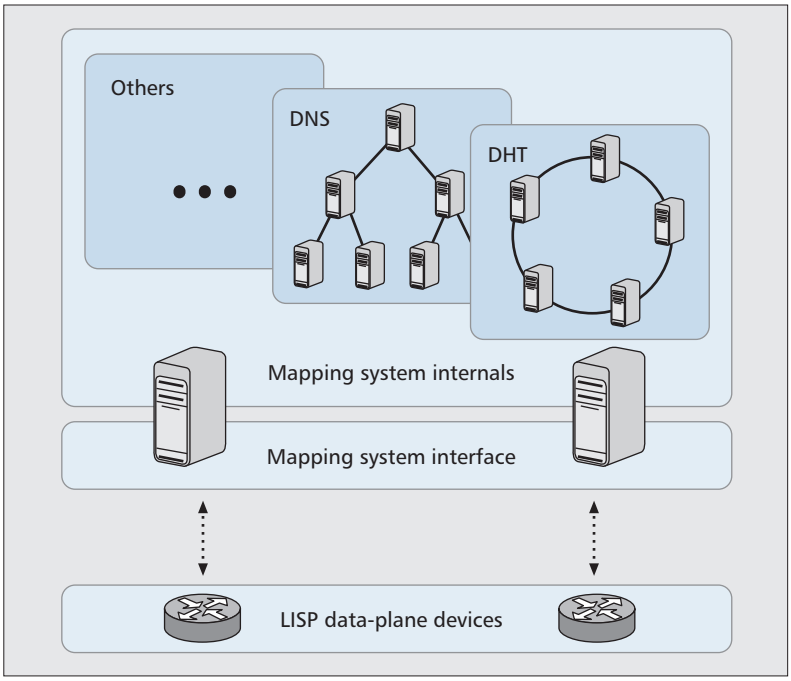

Figure 2. LISP mapping system.

the mapping system design used on the current LISP Internet deployment (lisp4.net). On the other hand, some deployments could require a flat name space; this is the case of non-aggregatable data such as character strings. For such requirements, a distributed hash table (DHT) design, rather than a DNS-like design, should be used. Although some initial efforts toward a DHT-like mapping system can be found in the literature [12], at the time of this writing only a hierarchical mapping system (DDT) has been successfully widely deployed.

Consistency: Levin *et al.* [7] expose the impact of a distributed SDN state on a logical centralized control application. While LISP still needs to deal with distributed trade-offs, its design allows mitigating them. The LISP mapping system is consistent and any snapshot of the distributed information reflects the desired control state. However, LISP network elements are eventually inconsistent, since an update on the mapping system is not instantaneously reflected on the data-plane. For instance, a LISP tunnel router can register new mapping information into the mapping system at any time, but an old version of the mapping can still be cached by remote tunnel routers. In order to minimize network inconsistency time, LISP defines two mechanisms to enforce up-to-date information at the data-plane. First, data-packets can carry an index of the current version of the mapping; second, a special control message can be used to explicitly notify remote parties of the mapping update.

Network Landmarks

Re-encapsulating tunnel routers (RTR) are special LISP tunnel routers that can be deployed on the RLOC space, rather than on the EID-RLOC edges. They receive LISP traffic, decapsulate it, look-up on the mapping system for the next hop,

re-encapsulate the traffic, and forward it. They give flexibility to the data path, offering network landmarks that data-packets can use.

These routers are a key element of a LISP SDN deployment. They can process the decapsulated traffic prior to re-encapsulating it again. This means, for instance, that traffic can be inspected, accounted, dropped, or modified at the re-encapsulating tunnel routers. An SDN approach can take advantage of these elements to set up network function devices. Devices such as firewalls, traffic analyzers, and accounting points, can be plugged, implemented, or virtualized on top of re-encapsulating devices. In that sense, Fig. 3 shows the abstract representation of a re-encapsulating tunnel router device with some network functions integrated that are used on demand.

Traffic Engineering

A mapping on the LISP mapping system can link an identifier to several locators. LISP allows defining a different priority and weight per locator. These values are used to specify the preference of the RLOCs to use to reach an EID as well as how to balance traffic among them. Besides that, LISP also introduces advanced traffic engineering capabilities by means of the explicit locator path (ELP). An explicit locator path is a list of hops through which packets have to be routed. The packets have to visit those locators in the same order as they are listed in the explicit path. These explicit paths serve as a mechanism to force traffic to follow a certain path on the locators space.

Priorities and weights also apply to locator paths, which means that an EID can map to several locator paths with different priority/weight attributes. Furthermore, such paths can be nested, creating sub-paths. This is done using EIDs instead of RLOCs as hops in the path. The final locator-only path will be obtained by a recursive look-up process. When a device finds that the next hop of the path is an EID, it will look-up on the mapping system to know the sub-path that this EID represents. Note that these sub-paths are subject to priority and weight values the same way as any other locator on the path. Using priority and weight, a LISP system can use different paths for the same destination where one path could be the most preferable while the others serve as backup, or many paths can be used at the same time to balance traffic.

Figure 4 shows an example. The traffic going to endpoint C should first go through M and N before being delivered at U. If that path is not available, then the traffic should be balanced in a 70/30 fashion over locators V and W. The traffic going to D should follow the path defined by α before reaching Z. As a backup, it can be also delivered directly on Z. In the example, α is used as a special identifier that represents a path instead of an endpoint.

Locator paths and re-encapsulating devices are tightly coupled, since in most of the cases the locator paths are used to force traffic to go through re-encapsulating devices. Explicit locator paths combined with re-encapsulating devices enable network programmability due to the ability to define custom programmable paths for packets in real time. Priority and weight parameters serve a fundamental role when deploying traffic rules. Traffic can be balanced among several paths and, thanks to recursion, to an arbitrary number of sub-paths. An SDN approach can deploy several re-encapsulating devices that also may implement (virtualized or not) network functions, and then program the mapping system to force the traffic to flow through these devices using explicit paths. Path nesting allows defining common sets of re-encapsulating devices that can be applied at once to specific traffic.

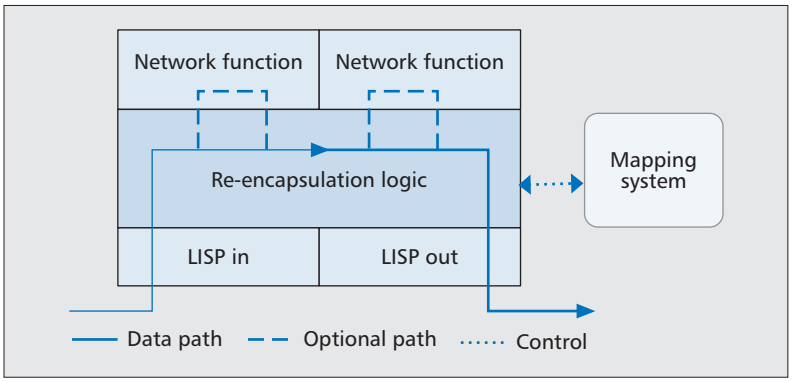

Figure 3. LISP re-encapsulating tunnel router.

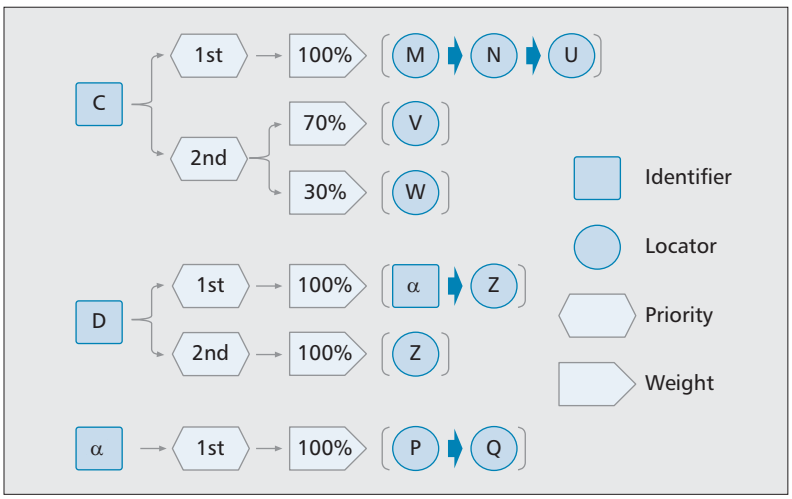

Figure 4. LISP traffic engineering.

Label System

Instance-ID is a 24 bit length identifier that can be associated with a certain EID. The identifier is included in the LISP header, and hence in all data-packets. Typically, this is used to carry VLAN tags or VPN identifiers. With this, network operators can split network policies and traffic, enabling multi-tenancy deployments.

However, instance-ID can be used beyond its original purpose. It is a 24 bit tag that can be appended to any data-packet to enable further features, not only multi-tenancy or address reusing. Specifically it can be used for routing scalability as well as management. In an SDN proposal such as Fabric [9], instance-ID can be used to tag flows that should be forwarded in the same way, simplifying forwarding on the core fabric and improving management and scalability.

LISP FOR SDN

Based on the previous analysis, this section discusses the advantages and drawbacks of applying LISP for SDN.

HIGHLIGHTS

Based on the analysis in the previous section, we highlight the most relevant features of LISP in SDN environments.

Scalability: As described previously, the mapping system provides scalability to the LISP system, an SDN solution can leverage this to provide a scalable network state database that can be directly queried by both data and control devices.

Interoperability: Given its flexible namespace and its label system, LISP is agnostic to the protocols it encapsulates and is well-suited to deploy overlays.

Inter-Domain: Network landmarks and LISP traffic engineering capabilities allow LISP to enforce policies on transit networks and make it suitable for inter-domain deployments.

BENEFITS

First, LISP has been designed to be incrementally deployable and to leverage current IP-based networks. Any existing IP-based network can incorporate common SDN features by simply upgrading some routers to LISP tunnel routers and connecting them to a mapping system.

Second, the shortcomings of traditional SDN protocols are motivating the emergence of hybrid SDN proposals that combine SDN with traditional network solutions [13]. Interestingly, due to its scalability and interoperability, LISP eases the deployment of the aforementioned hybrid SDN networks, specially since LISP can be incrementally deployed. Furthermore, thanks to its flexibility, LISP is well-suited to accommodate future protocols and new network approaches.

Finally, in contrast with common SDN protocols that are designed to operate mostly within a single domain, LISP allows SDN policies to be enforced across domains (e.g. DC-to-DC, DC-to-user's home). Well-placed LISP elements (e.g. re-encapsulating tunnel routers) make possible a programmable SDN deployment over a transit network (e.g. the Internet), something that is more complex to accomplish with traditional SDN protocols.

DRAWBACKS

Due to both how the protocol operates and its nature as a map-and-encap approach, LISP has some limitations that must be taken into account when considering LISP as a southbound SDN protocol.

Extra Headers: In order to encapsulate the traffic, LISP adds extra headers to the packets. This increments the packet size and reduces the available payload.

Mapping Resolution: LISP devices resolve and cache the mapping information on demand. The first packets of non-cached flows need to be either buffered or dropped until the mapping resolution process has been completed.

Mapping Updates: Any update on the mapping system is propagated over the network. However, this propagation involves some delay due to the signaling process, which can introduce latency in the system and/or produce packet losses.

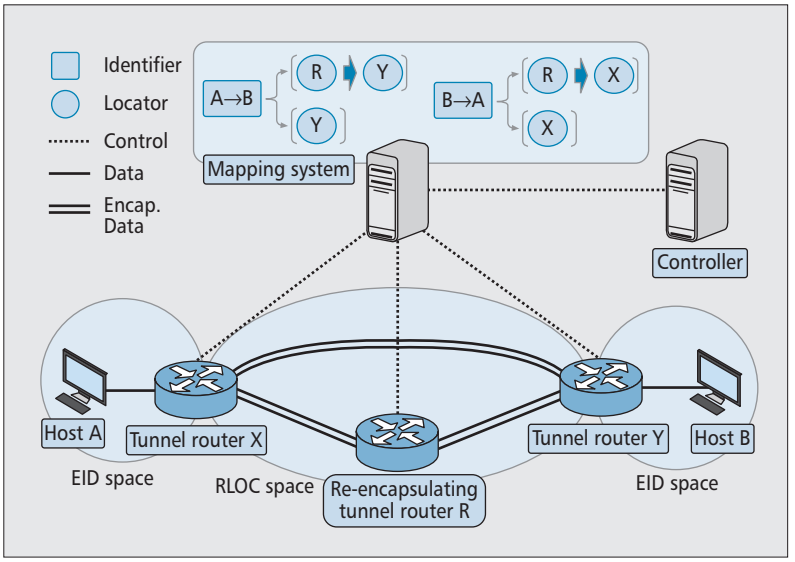

Figure 5. LISP SDN prototype.

Look-Up Support: While LISP defines how to convey different types of addresses in control messages, it does not define how to use all of those addresses to perform look-up operations.

Flat Data Support: Generally, mapping system implementations have been designed with hierarchical data in mind (e.g. IP addresses) and as such do not perform well when storing flat data (e.g. character strings).

Both *extra headers* and *mapping resolution* drawbacks are inherent in the LISP architecture. However, they do not have a strong impact on performance given that LISP encapsulation typically adds only 36 bytes (IPv4) or 56 bytes (IPv6) [1], and the LISP entities cache the mappings and because of the strong locality of traffic [14] achieve a hit-rate above 99 percent.

Regarding *flat data support*, the limitation can be solved with a DHT-based mapping system. Given that the interface to read/write mappings is open and standard, this limitation is not architectural and can be solved taking advantage of existing DHT databases.

To overcome the rest of the drawbacks we propose the following potential enhancements for the protocol above.

PROPOSED IMPROVEMENTS

The mapping updates limitation requires optimizing the mapping update signaling on SDN scenarios. In this context, we propose implementing a publish/subscribe mechanism for LISP mappings. The proposed mechanism is already being prototyped for the LISP project in OpenDaylight (opendaylight.org). The system operates as follows. Whenever a LISP data-plane device requests a mapping, the mapping system adds it to the list of subscribers for that mapping. Whenever the mapping data changes, the subscribers of that mapping are immediately notified, and thus they do not need to wait for the standard mapping update

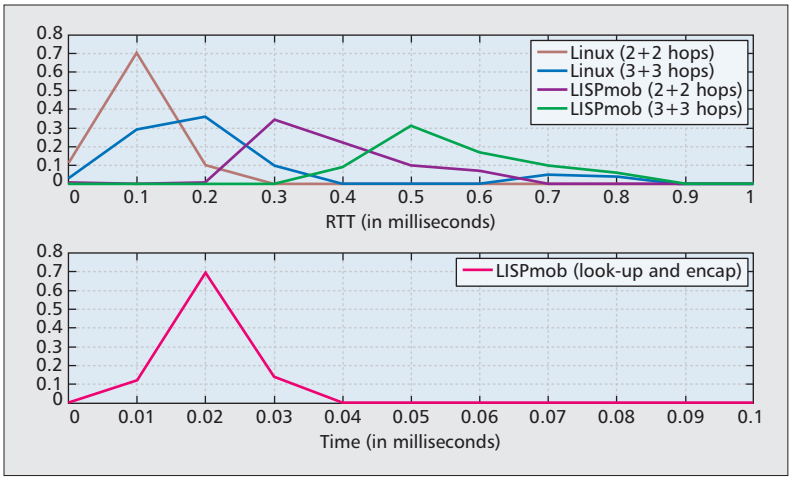

Figure 6. LISPmob induced delay.

propagation. The requester has to renew its subscription by explicitly requesting the mapping before a time-out. For scenarios where scalability and/or security is a concern, the subscription may be restricted to a set of predefined mappings or subscribers.

The *look-up support* needs to be extended beyond its current focus mostly in IP-prefixes. Most of the current SDN solutions operate the network in terms of flows. Traditionally, the minimal amount of information to identify a flow is its 5-tuple, even though normally in SDN more fields are used (e.g. OpenFlow). We advocate that LISP requires at least a look-up mechanism based on 5-tuples, despite the fact that in the future further look-up processes can be implemented, potentially leveraging on the OpenFlow tuple matching process.

Prototype

This section presents a prototype of a LISP-based SDN solution in order to validate its feasibility.

Setup

The prototype topology is depicted in Fig. 5. Two hosts (A and B) in different LISP sites are connected through the transit network via two tunnel routers (X and Y) and optionally via a re-encapsulating tunnel router. The mapping system stores mappings of source-destination EID tuples to RLOC space paths. These mappings are loaded by (borrowing OpenFlow terminology) a controller.

To implement the prototype, we instantiate a virtual machine running Linux for each of the elements on the topology. We connect the machines using virtual networks, emulating the topology depicted in the figure. On the machines that need LISP capabilities we run the open-source LISPmob implementation (lispmob.org) modified to support look-ups based on destination-source EID tuples.

On the described prototype we test two different scenarios, one where the traffic goes directly to its destination and another where the traffic goes through the detour introduced by the re-encapsulating tunnel router. We have a simple SDN application running on the controller that can dynamically set which path has higher priority.

Metrics

To extract relevant metrics we run 10 iterations injecting ping packet traffic during 10 secs per scenario (with and without detour) at a data-rate of 1000 pkts/s.

Packet Loss: The initial packet loss is due to the required mapping resolution signaling when we send a flow over a new path. We have measured an average initial packet loss of 3.0 packets dropped per iteration on the scenario without detour. This average packet loss goes up to 5.1 packets per iteration on the scenario with the extra LISP router due to the introduction of additional mapping resolution operations.

Delay: To measure how much delay is introduced by LISPmob we built an equivalent prototype without LISP capabilities, where traffic paths were configured modifying the routing tables on the Linux boxes. The top of Fig. 6 shows the PDF (probability distribution function) of the RTT (round trip time) for the scenarios considered. Note that without the detour round-trip traffic goes through four hops (i.e. two from A to B and two from B to A), while the detour introduces one extra hop in each direction (3+3) for a total of six hops. The bottom part of Fig. 6 shows how much time elapses from when LISPmob receives a new packet until it delivers the LISP encapsulated packet, that is, LISP look-up and encapsulation. The plots in Fig. 6 show that each LISP hop adds roughly 50 microseconds to the RTT, of which no more than 30 are due to LISP operations. The remaining latency is mostly due to the user ↔ kernel communication required by LISPmob. Nevertheless, the lookup and encapsulation operations may be optimized by router manufacturers to enable the performance of hardware implementations to be similar to that of traditional IP datagram forwarding.

Conclusions

In this article we have analyzed if LISP, in its current form, can be used for SDN. Our analysis concludes that the control-data decoupling, the network programability, and the centralized control enabled by traditional SDN solutions are already enabled by the LISP mapping system and supported by the rest of the LISP components. The major benefits of using LISP for SDN are that it keeps its incremental deployability and flexibility while providing scalability, interoperability, and inter-domain support, making LISP especially suitable for SDN deployments over legacy or transit networks, such as the Internet. However, despite its potential as an SDN enabler, there are some aspects of the protocol that should be extended to better fit the SDN use-case, mainly the signaling for the mapping updates and implementing support for an advanced look-up process. Finally, the presented prototype demonstrates that LISP is feasible for SDN scenarios.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. This work has been partially supported by a Cisco research grant, by the Spanish Ministry of Education under grant FPU2012/01137, by the Spanish Ministry of Economy and Competitiveness under grant TEC2014-59583-C2-2-R, and by the Catalan Government under grant 2014SGR-1427.

REFERENCES

[1] D. Farinacci *et al.*, "The Locator/ID Separation Protocol (LISP)," IETF RFC 6830, 2013.
[2] M. Jarsche *et al.*, "Interfaces, Attributes, and Use Cases: A Compass for SDN," *IEEE Commun. Mag.*, vol. 52, no. 6, June 2014, pp. 210–17.
[3] N. Foster *et al.*, "Languages for Software-Defined Networks," *IEEE Commun. Mag.*, vol. 51, no. 2, 2013, pp. 128-134.
[4] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, 2013, pp. 114–19.
[5] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, 2008, pp. 69–74.
[6] S. Sezer *et al.*, "Are We Ready for SDN? Implementation Challenges for Software-Defined Networks," *IEEE Commun. Mag.*, vol. 51, no. 7, 2013, pp. 36–43.
[7] D. Levin *et al.*, "Logically Centralized?: State Distribution Trade-Offs in Software Defined Networks," *Proc. First Workshop on Hot Topics in Software Defined Networks*, ACM, 2012, pp. 1–6.
[8] S. H. Yeganeh *et al.*, "On Scalability of Software-Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, 2013, pp. 136–41.
[9] M. Casado *et al.*, "Fabric: A Retrospective on Evolving SDN," *Proc. First Workshop on Hot Topics in Software Defined Networks*, ACM, 2012, pp. 85–90.
[10] T. Koponen *et al.*, "Onix: A Distributed Control Platform for Large-Scale Production Networks," *OSDI*, vol. 10, 2010, pp. 1–6.
[11] L. Jakab *et al.*, "LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System," *IEEE JSAC*, vol. 28, no. 8, 2010, pp. 1332–43.
[12] L. Mathy and L. Iannone, "LISP-DHT: Towards a DHT to Map Identifiers onto Locators," *Proc. 2008 ACM CoNEXT Conf.*, ACM, 2008.
[13] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and Research Challenges Of Hybrid Software Defined Networks," *SIGCOMM Comput. Commun. Rev.*, 44, 2, April 2014, pp. 70–75.
[14] F. Coras, A. Cabellos-Aparicio, and J. Domingo-Pascual, "An Analytical Model for the LISP Cache Size," *Proc. IFIP Networking*, 2012.

BIOGRAPHIES

ALBERTO RODRIGUEZ-NATAL received a BSc. (2010) and a MSc. (2012) in computer science from the University of Leon (Spain) and the Technical University of Catalonia (Spain), respectively. He is now a Ph.D. candidate at the Technical University of Catalonia and has been a visiting researcher at Cisco Systems (USA) and the National Institute of Informatics (Japan). His main research interests are future Internet architectures and Software-Defined Networking.

MARC PORTOLES-COMERAS received his degree in telecommunications engineering from the Technical University of Catalonia (UPC) and is currently working as a software engineer at Cisco Systems Inc., participating in the development of the LISP protocol architecture. Before joining Cisco he was a research engineer at the Centre Tecnológic de Telecomunicacions de Catalunya (CTTC) where he participated in multiple R&D projects. His current research interests are SDN and network virtualization solutions.

VINA ERMAGAN is a technical lead in the Chief Technology and Architecture Office at Cisco Systems. She joined Cisco in 2008 and has been working on research, design, and development of SDN and network virtualization technologies ever since. She has initiated projects to implement LISP in Open vSwitch (OVS), OpenStack, and OpenDaylight. Vina received her MSc. in computer science from UC San Diego in 2008, and her BSc. in computer engineering from Sharif University of Technology.

DARREL LEWIS has more than 25 years of experience as an engineer for routing infrastructure vendors and network service providers. He has co-authored several LISP RFCs and he is currently a technical leader at Cisco Systems. Previously, he worked for Riverhead Networks as the lead consulting engineer. He is active in the North American Network Operators Group (NANOG), the Internet Engineering Task Force (IETF), and is a noted instructor in the fields of both Internet routing and security.

DINO FARINACCI is a technologist advancing the state of the art for the next-generation Internet. He was the original co-author for LISP dating back to 2007 and has the pleasure of writing two implementations of the protocol. He currently does consulting for large and startup networking vendors as well as users of such products. He is a software engineer by trade and a technology visionary by passion.

FABIO MAINO is a distinguished engineer at Cisco Systems, in the Chief of Technology and Architecture Office, where he leads the LISP research team. He has approximately 50 patents issued or filed with the US PTO, and has contributed to various standardization bodies, including IEEE, IETF, and INCITS. He has a Ph.D. in computer and network security and an M.S. ("Laurea") in electronic engineering from Politecnico di Torino, Italy.

ALBERT CABELLOS-APARICIO received a BSc. (2001), MSc. (2005), and Ph.D. (2008) degree in computer science from the Technical University of Catalonia (UPC), where he is now an assistant professor. In 2010 he joined the NaNoNetworking Center in Catalunya, where he is the Scientific Director. He has co-authored more than 15 journal and 40 conference papers. His main research interests are future architectures for the Internet and nano-scale communications.