Programmable Overlays via OpenOverlayRouter

Alberto Rodriguez-Natal, Jordi Paillisse, Florin Coras, Albert Lopez-Bresco, Lorand Jakab, Marc Portoles-Comeras, Preethi Natarajan, Vina Ermagan, David Meyer, Dino Farinacci, Fabio Maino, Albert Cabellos-Aparicio

***Abstract*—OpenOverlayRouter (OOR) is an open-source software router to deploy programmable overlay networks. OOR leverages the Locator/ID Separation Protocol (LISP) to map overlay identifiers to underlay locators, and to dynamically tunnel overlay traffic through the underlay network. LISP overlay state exchange is complemented with NETCONF remote configuration and VXLAN-GPE encapsulation. OOR aims to offer a flexible, portable, and extensible overlay solution via a user-space implementation available for multiple platforms (Linux, Android, and OpenWrt). In this article, we describe the OOR software architecture and how it overcomes the challenges associated with a user-space LISP implementation. Furthermore, we present an experimental evaluation of OOR performance in relevant scenarios.**

I. Introduction

OVERLAY networks have been used over the years to circumvent the constraints of physical networks. Overlays allow bypassing the limitations of current deployments and enhancing networking infrastructure without replacing the hardware already in-place. These networks have proved to be useful for a broad range of use-cases, such as multicast [1], traffic engineering [2], resilient networks [3] or peer-to-peer networking [4]. Furthermore, with the advent of Software-Defined Networking (SDN), overlays have become a tool to enable SDN capabilities over legacy network equipment [5], [6].

Among the different options to instantiate overlays, the Locator/ID Separation Protocol (LISP) [7] has gained significant traction among industry and academia [5], [6], [8], [9], [10], [11], [14], [15]. Interestingly, LISP offers a standard, inter-domain, and dynamic overlay that enables low-CAPEX innovation at the network layer [8]. LISP follows a map-and-encap approach where overlay identifiers are mapped to underlay locators. Overlay traffic is encapsulated into locator-based packets and routed through the underlay. LISP leverages a public database to store overlay-to-underlay mappings and on a pull mechanism to retrieve those mappings on demand from the data-plane. Therefore, LISP effectively decouples control and data planes, since control-plane policies are pushed to the database rather than to the data-plane. Forwarding elements reflect control policies on the data-plane by pulling them from the database. In that sense, LISP can be used as an SDN southbound protocol to enable programmable overlay networks [5].

In this article we present OpenOverlayRouter[1] (OOR), a community-driven project focused on developing an open-source software router to deploy LISP-based overlays. Open-sourced under an Apache 2.0 license, OOR can run on Linux computers, Android devices, and OpenWrt[2] home routers. OOR supports both LISP [7] and LISP-MN [9] (a lightweight version of LISP intended for mobile devices) for overlay state exchange, NETCONF (RFC 6241) protocol for remote management and configuration, and LISP & VXLAN-GPE[3] formats for encapsulation. OOR is a renaming of the LISPmob[4] project, the original implementation of the LISP-MN specification, after LISPmob grew in features and capabilities over the years. From a minimal LISP-MN implementation, LISPmob evolved into a complete LISP implementation, and from there to a comprehensive overlay solution that incorporates other protocols beyond LISP, effectively becoming OpenOverlayRouter.

OOR's focus is on enabling network programmability at the edge, with special interest into providing added value to end-users. To that end, OOR's code runs entirely at the Linux user-space and has a common code-base for all supported platforms. This software approach allows the OOR community to have a strong focus on flexibility, customization, and development of new features. In this context, OOR represents a solid base for research, innovation, and prototyping of new overlay use-cases. An example of OOR's success is that it is being used by LISP projects in major SDN controllers, i.e. LispFlowMapping[5] in OpenDayLight[6] and SBI LISP [10] in ONOS[7].

In what follows, we describe OOR's software architecture and components. We first give an overview of the main architectural core ideas behind its implementation to later delve into the internals of its architecture (both control and data plane). In addition, we present the benefits of instantiating overlays via OOR, describe successful use-cases across the OOR community and compare OOR to other similar software

A. Rodriguez-Natal, F. Coras, L. Jakab, M. Portoles-Comeras, P. Natarajan, V. Ermagan and F. Maino are with Cisco Systems, San Jose, CA, USA. E-mail: {albrodr2, fcoras, lojakab, mportole, prenatar, vermagan, fmaino}@cisco.com

J. Paillisse, A. Lopez-Bresco and A. Cabellos-Aparicio are with Technical University of Catalonia, Barcelona, Spain. E-mail: {jordip, alopez, acabello}@ac.upc.edu

D. Meyer is with Brocade Communication Systems, San Jose, CA, USA. E-mail: dmm@1-4-5.net

D. Farinacci is with lispers.net, San Jose, CA, USA. E-mail: farinacci@gmail.com

[1] http://openoverlayrouter.org [Accessed on 13th March 2017]

[2] http://openwrt.org [Accessed on 13th March 2017]

[3] https://tools.ietf.org/html/draft-ietf-nvo3-vxlan-gpe-03 [Accessed on 13th March 2017]

[4] http://openoverlayrouter.org/lispmob [Accessed on 13th March 2017]

[5] http://docs.opendaylight.org/en/stable-boron/user-guide/lisp-flow-mapping-user-guide.html [Accessed on 13th March 2017]

[6] https://opendaylight.org [Accessed on 13th March 2017]

[7] http://onosproject.org/ [Accessed on 13th March 2017]

 DOI: 10.1109/MCOM.2017.1601056

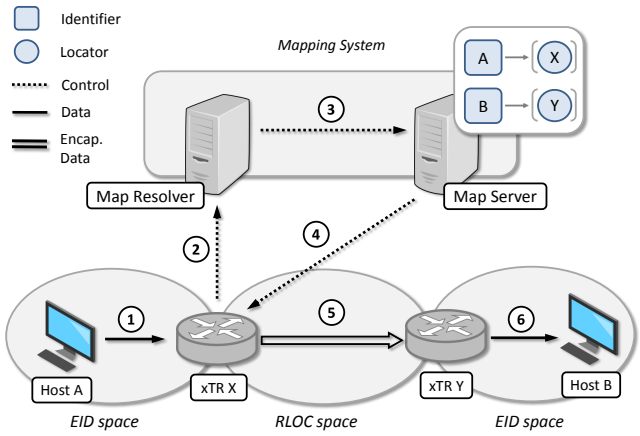

Fig. 1. **LISP Overview.**

solutions. Finally, we present an experimental evaluation of OOR in relevant scenarios and compare its performance with related implementations. As results show, although taking a user-space approach, OOR's implementation results in a remarkable performance suitable for home and edge devices.

II. LISP Background

The Locator/ID Separation Protocol (LISP) instantiates overlays via decoupling host identity from its location. It creates two different namespaces: Endpoint IDentifiers (EIDs) and Routing LOCators (RLOCs). Each host is identified by an EID, and its 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 more address families to be used as well.

Packets are routed based on EIDs at LISP sites, and on RLOCs at transit networks. At LISP sites edge points, Ingress/Egress Tunnel Routers (xTR) are deployed to allow transit between EID and RLOC space. To do so, LISP follows a map-and-encap approach. EIDs are mapped to RLOCs and the xTRs encapsulate EID packets into RLOC traffic. LISP introduces a publicly accessible Mapping System, which is a distributed database containing EID-to-RLOC mappings. The Mapping System is composed of Map-Resolvers (MR) and Map-Servers (MS). Map-Servers store mapping information, and Map-Resolvers find the Map-Server storing a specific mapping. Fig 1 shows an example of LISP workflow. Host A wants to communicate (1) with its peer B, from which it only knows its EID. xTR X sends (2) a Map-Request to obtain the RLOC of the xTR serving host B. This Map-Request is routed (3) through the Mapping System to finally reach the Map-Server containing this info. The Map-Server replies (4) to xTR X with a Map-Reply message. With the mapping information, xTR X is able to encapsulate and send (5) the data packet to xTR Y which decapsulates and forwards it (6) to peer B. Other relevant LISP devices are: Proxy xTRs (PxTR), that can be used to connect to legacy (i.e. non-LISP) sites, Re-Encapsulating Tunnel Routers (RTR), to enforce in-path policies, and LISP Mobile Nodes (LISP-MN). In LISP-MN the xTR is embedded within the mobile node, and connections -at transport level- are preserved across handover events.

In terms of history, LISP was initially created as an outcome of a 2006 Internet Architecture Board Workshop (RFC4984) that concluded that *the most important problem facing the Internet today* was the continued growth of the Border Gateway Protocol (BGP) routing tables in the default-free zone. This resulted in a plethora of solutions to address this issue where, among them, LISP gained a lot of traction. With LISP, EIDs are allocated to sites in a provider-independent manner, but they are not advertised in the global Internet. The global BGP routing tables would eventually only contain RLOCs; then it would be possible for these to be assigned in such a way that transit network providers could highly aggregate them, and help scale the BGP routing tables. Although this was the initial goal, the dynamic mapping of EID to RLOCs inherently enables programmable overlays. As a result, LISP has been applied to many other use-cases beyond the scalability of the default-free zone. Particularly interesting areas are SDN [5] and NFV [6], as well as edge overlays, which are mainly covered by OOR.

III. Architecture Overview

OOR is an open-source implementation of both LISP [7] and LISP-MN [9], written in C for Linux-flavored systems. The main goal of OOR is to represent a solid code-base for overlay research, innovation, and prototyping with focus on offering easy programmability and end-user support. To achieve this, OOR comprises a modular architecture with a user-space approach and a multi-platform implementation.

A. Modular Design

Although it runs as a single user-space daemon, internally OOR is composed of different software modules. The modules have been abstracted and their interactions well-defined. This allows different components of OOR to be inspected and modified without disrupting the rest of the system. An overview of the different modules is depicted in Fig. 2. There are two main modules, *control* and *data* which support the control and data planes respectively. The *data* module handles data packets processing while the *control* module keeps control state, regulates signaling, and manages mapping information used by the *data* module. It should be noted that OOR implements several LISP devices. Each device is represented with a module that adapts the *control* and *data* modules behavior in order to match with the specific LISP device. Beyond those main modules, certain parts of OOR have been abstracted into auxiliary modules that connect to the main ones, such as interface management, multihoming procedures or database storage.

B. User-Space Implementation

All OOR code runs in user-space. This approach presents many advantages. First, it prevents having to delve into hardware specific optimizations and avoids the complexity and

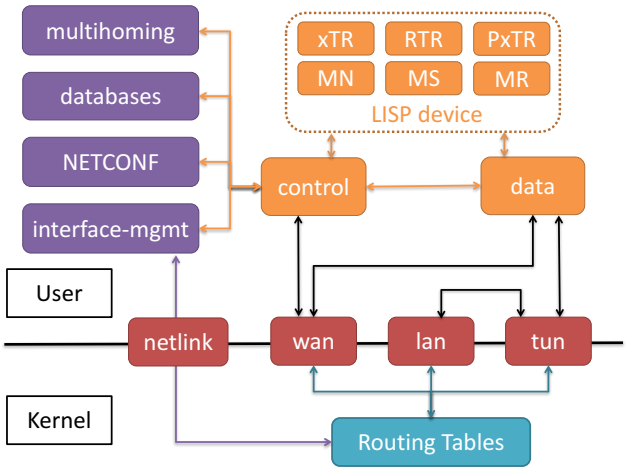

Fig. 2. **OOR architecture**

high maintenance costs typically associated with kernel code. Second, software can be easily ported to other platforms (see section III-C) making it available in a wide range of devices. Third, it lowers the entry barrier for new contributors to the project. On the contrary, the main drawback of user-space implementations is the performance drop due to communication between kernel and user spaces. However, achieving high throughput is not the primary goal of OOR, since it is targeted at the network edge (eg. mobile nodes, home routers), where devices typically do not require very high bandwidth.

In order to support a LISP data-plane on user-space, OOR uses TUN/TAP[8] drivers. Specifically, it creates a TUN interface to capture and forward traffic. TUN virtual devices allow user-space applications to receive and transmit network layer packets. Figure 2 depicts OOR components in user-space and how they interact with kernel space. The *data* module hooks to the TUN interface for data processing while the *control* module opens a socket on the WAN interface to control signaling. Finally, OOR uses netlink to monitor and modify the routing tables.

C. Multi-Platform

Thanks to its modular architecture and its user-space approach, OOR supports three different platforms while using the same code base. The Linux implementation of OOR has been ported to OpenWrt (home routers) and Android (mobile devices). Leveraging these three different flavors, OOR users have reported successful deployments of OOR on desktops, laptops, routers, and smartphones, as well as on Raspberry Pi[9] devices and Arduino[10] boards.

IV. Control-Plane Components

This section describes relevant implementation details of the OOR *control* module. OOR supports different LISP devices via a unified *control* module that accepts pluggable device-specific modules (see Fig. 2). The control module is configured using a static configuration file which is read during boot-up. At run-time, it is possible to modify parts of this configuration remotely thanks to the NETCONF (RFC 6241) module that OOR implements with the *libnetconf* library[11]. Other major auxiliary modules are described in this section.

A. Databases

OOR uses two different EID-to-RLOC mapping databases. One is to store local mappings (e.g. EID prefixes being served by an OOR xTR) that are configured during bootstrap or via NETCONF. The other one, usually referred to as the *map-cache*, stores mappings learned when pulling information from the Mapping System (e.g. EID prefixes served by remote xTRs). Both databases are implemented over Patricia Trie structures kept in memory. Patricia Tries are a special case of Radix Tries where the radix equals 2. In other words, a Patricia Trie is an optimized version of a digital tree where single-child nodes are merged with their parents. This generates an optimal data storage for strings that share long prefixes, such as the bit-strings of IP addresses. Since LISP mappings are (generally) indexed based on IP prefixes, Patricia Trie databases allow OOR to optimally store such indexes and retrieve the most specific prefix for a given IP address. However, OOR's modularity supports as well unplugging the Patricia-based structures and using other databases for non-IP based mappings.

B. Multihoming

In multihoming scenarios -where an xTR has several locators available at the same time- users have to define inbound and outbound traffic policies. This is done in LISP by configuring priorities and weights for the available locators. Following the LISP specification, OOR does not load-balance traffic per packet but rather per flow (defined as a sequence of packets identified by the same 5-tuple). This approach avoids splitting flows over different paths that may have different delay/jitter and hence, may severely impact on performance. In order to load-balance traffic according to the configured weights (for locators that have the same priority), OOR uses a vector that assigns positions to locators based on their weight (e.g., if two locators have weights 10 and 15 the module will assign the 40% of the vector positions to the first one and the 60% to the second). Flows are then forwarded to (and through) locators based on randomly chosen vector positions.

C. Interface Management

In order to manage system interfaces, OOR opens a netlink socket to the kernel, which is used to modify routing tables (see section V) as well as to monitor changes in these tables or in the network interfaces. The events currently filtered and processed are: interface status up, interface status down, new

[8]http://www.kernel.org/doc/Documentation/networking/tuntap.txt [Accessed on 13th March 2017]

[9]https://www.raspberrypi.org/ [Accessed on 13th March 2017]

[10]https://www.arduino.cc/ [Accessed on 13th March 2017]

[11]https://github.com/CESNET/libnetconf [Accessed on 13th March 2017]

IP address assigned to an interface, and new entries in the routing tables. Such events are processed as follows:

When OOR detects a new IP address assigned to an interface it updates its internal structures. If needed, it also updates the Mapping System information and the cached information on remote peers through LISP control signaling. In the event of an interface going up or down, it follows the same procedure, but it also checks if its multihoming state is still valid. This is due to the fact that in some cases new locators are available or previously available locators are no longer usable. Finally, if OOR detects a new entry on the routing tables it checks if there is a new gateway for any of the interfaces it is monitoring and, if required, updates the routing tables to handle outgoing RLOC packets (see section V).

V. User-Space Data-Plane

This section presents how OOR implements the data-plane of the different LISP devices. The OOR data-plane is implemented in the *data* module, which is responsible for encapsulating and decapsulating data packets. Although some LISP devices do not need data-plane (Map-Server/Map-Resolver), for the remaining devices (xTR, RTR, and PxTR) data-plane operation is quite similar; only Mobile Nodes (MN) require a slightly different approach. Regarding address families, OOR supports only IPv4 and IPv6 for both EIDs and RLOCs, even though LISP allows other address families to be used. At the time of this writing, support for L2 addresses (e.g. Ethernet MAC) is on OOR's roadmap. Finally, OOR is agnostic to the specific encapsulation format and complies with both LISP and VXLAN-GPE specifications. VXLAN-GPE headers are binary compatible with LISP headers and thus support for the former is almost immediate once the latter is implemented.

Regular OOR data-plane runs in user-space on top of the Linux kernel networking stack. However, there is an ongoing effort to make OOR compatible also with Vector Packet Processor (VPP)[12]. VPP is an optimized data-plane that bypasses the kernel stack and offers high performance. We leave the analysis and measurements of such approach for future work.

A. Encapsulation via TUN

When processing outgoing traffic (i.e. from EID space to RLOC space), OOR needs to capture EID space traffic, encapsulate, and forward it. In order to intercept outgoing EID traffic (both on xTR and PxTR modes), OOR redirects it to the TUN interface and retrieves it. This redirection is achieved modifying the Linux routing tables and routing rules. In xTR mode, since local traffic does not have to be encapsulated, the new routes and rules forward to TUN all outgoing traffic from the EID space that is not addressed to the local EID space itself. RTRs operate on RLOC space and hence they receive EID traffic encapsulated directly from an RLOC interface.

Based on the EID traffic, OOR builds outer headers using RLOC addresses (governed by the *control* module). To speed-up processing time, the *data* module keeps a hash table with information from already processed packets to avoid querying the *control* module on a per-packet basis. OOR writes the encapsulated traffic into a socket, which injects traffic again in the Linux routing system.

In multihomed scenarios with several default routes, OOR must ensure that Linux chooses the appropriate outbound interface. To achieve this, OOR creates (for each RLOC interface) a table that only includes a route to the gateway of the interface and, in turn, for each table a rule that matches packets using that particular source RLOC.

To manage incoming traffic (i.e. from the RLOC space to the EID space) OOR opens a socket listening for encapsulated traffic. Received RLOC traffic is decapsulated and written in the TUN interface, then the kernel forwards EID packets to the EID space. In RTR mode traffic is re-encapsulated with new RLOC headers and forwarded back to the RLOC space.

B. Mobile Node Considerations

Although a LISP-MN operates fundamentally as an xTR, additional considerations must be taken into account. The major difference between an xTR and a MN is that a LISP tunnel router (xTR) receives packets from an external source, while in a LISP-MN such packets are generated -by the applications running- in the device itself. In mobile node operation, the TUN interface must be provisioned with the mobile node EID address. Applications running on the MN bind sockets to this interface and use its EID as source address. In order to enforce that all the applications bind to the TUN interface, OOR configures it as the most preferable route for non-local traffic. Additionally, it configures specific tables and rules per each RLOC interface and therefore, once encapsulated, traffic will be forwarded to the correct outgoing interface, thus effectively preventing loops.

To allow correct packet reception in Linux, OOR deactivates reverse path forwarding (RPF) verifying mechanisms to prevent discarding packets. In Linux, RPF works as follows: for every received packet the kernel checks -according to its routing tables- the output interface for that particular source address. If the input interface is different from the output interface, the RPF mechanism discards the packet as an anti-spoofing mechanism. While OOR is running, Linux detects that any non-local destination address is reached through the TUN interface, thus RLOC [packets arriving via a physical interface would not pass the RPF check.

VI. Discussion

OOR as an overlay solution presents a set of benefits and drawbacks, as well as different levels of applicability to different use-cases. In this section, we discuss the pros and cons of using OOR to instantiate overlays and summarize several relevant use-cases that the OOR community has found over the years.

A. Overlays via OOR

Overlays have been used since the early days of the Internet for a wide variety of applications, such as enabling multicast

[12]https://fd.io/technology [Accessed on 13th March 2017]

[1], improving delay through overlay routing [2], making networks resilient [3] or instantiating peer-to-peer networks [4]. Unfortunately, overlays are typically static, i.e., they are setup once and rarely modified nor reprogrammed. However, LISP and OOR can provide fully programmable dynamic overlays. The main advantage of OOR overlays is that they follow a pull-based approach. The state is not statically provisioned, but instead programmed on a central entity (i.e the Mapping System) and requested on-demand by data-plane elements. In addition, thanks to OOR, LISP overlays can be effectively instantiated on the very edge of the network and over heterogeneous devices (Android phones, OpenWrt routers, Linux servers, etc). On the other hand, a major disadvantage of OOR overlays (and LISP overlays in general) is that they require encapsulating packets, and thus they introduce some overhead. Moreover, there may be an initial packet loss when the LISP overlay state is not ready, since signaling mechanisms must pull the state from the Mapping System to be able to forward subsequent packets. Finally, while it provides several benefits, the user-space approach of OOR limits the data-plane throughput that can be achieved.

B. Use-Cases for OOR

Over the years, the OOR community has reported different success stories of real-world use-cases of OOR. For instance, several users have been able to leverage OOR as a way to bypass their IPv4-only providers, in order to gain connectivity with publicly reachable IPv6 addresses. In other cases, people have used OOR as a way to keep a fixed IP address across handover events while avoiding triangular routing (e.g. for mobile servers). However, one of the major success stories is the use of OOR for easy home multihoming.

Multihoming offers important advantages for users since they can connect to the Internet through several providers at the same time, eliminating provider lock-in, and enabling bandwidth aggregation while potentially reducing costs (combining several low-speed connections may be cheaper than using one high-speed connection). Still, multihoming is typically only available to large BGP-capable networks.

OOR enables end-users to deploy multihoming in small routers (homes or small offices). The OOR community offers a fully pre-configured OpenWrt binary installation file for end-users. Thanks to the LISP Beta Network[13] (an experimental LISP network currently operated by OOR maintainers), users are provided an EID and the required LISP infrastructure to connect both to LISP and non-LISP sites. The interested reader can find more information about this in the OOR wiki[14]. This particular use-case has been highlighted in printed press[15], which has drawn further attention to OOR.

Finally, OOR is playing an important role in research. Currently, there are other academic initiatives (e.g., [11]) that are making use of the project for their own research. Furthermore, OOR has helped to discover new research challenges that need to be addressed. A notable example can be found, for instance, in the map-cache [12].

[13]http://www.lisp4.net/beta-network/ [Accessed on 13th March 2017]

[14]https://github.com/OpenOverlayRouter/oor/wiki/Easy-Multihoming [Accessed on 13th March 2017]

[15]https://www.heise.de/ct/ausgabe/2017-3-LISP-auf-Fritzboxen-OpenWRT-und-Cisco-IOS-3595908.html [Accessed on 13th March 2017]

VII. Related Work

Recent software solutions provide similar capabilities to those offered by OOR while having a different architectural approach. First, FlowTags [13] provides dynamic policy enforcement over the network, specially tailored for middleboxes, by including tags in packets. Second, EMPOWER[16] provides a Network Operating System to create network slices with an emphasis on edge networks and third, the jFED[17] framework helps researchers create network experiments regardless of the underlying topology, also allowing edge deployments. Finally, with a similar architecture to OOR, OpenVPN[18] offers user-space packet encapsulation with a static pre-loaded configuration.

In addition, OOR is not the only software implementation capable of enabling LISP overlays. OpenLISP [14] is a BSD kernel LISP implementation and jLISP [15] is an open-source Java implementation with a focus on portability and extensibility. However, to the best of our knowledge, OOR is the only available solution that brings dynamic LISP overlays to the very edge of the network and enables LISP overlay capabilities on end-user devices.

VIII. Evaluation

This section presents an experimental evaluation of the performance of OOR. To the best of our knowledge OOR is the only mature LISP implementation that takes a full user-space approach and thus, there are no reference implementations to compare with. Instead and when relevant, we compare OOR performance with OpenLISP and OpenVPN.

A. Throughput

Here we focus on the throughput of OOR's user-space data-plane. OOR is compiled for Linux and installed in two Intel Core 2 PCs (3GHz, 4GB RAM) running Ubuntu 14.04, both machines are connected over a dedicated Gigabit Ethernet link. OpenLISP 2.0.2 runs on the same machines with FreeBSD 9.2. Traffic is generated using the *nuttcp* tool (UDP packets of 1388 bytes), and we monitor both input and output rates. As figure 3 shows, OOR scales close to the link capacity with a maximum throughput of 800Mbps, at this rate the OOR user-space process is using all the available CPU. OpenLISP with its kernel implementation is very close to link capacity.

We also measure the throughput of the Android and OpenWrt OOR implementations and compare it to OpenVPN (1.1.14 for Android, 2.2.2 for OpenWrt) on which, for the fairness of the comparison, we deactivate encryption and configure UDP traffic. In Android we generate traffic using *iperf* running on a Nexus 7 (Android 4.3) over a WiFi link (802.11g), for OpenWrt we run OOR on a Netgear home router

[16]http://empower.create-net.org/ [Accessed on 13th March 2017]

[17]http://jfed.iminds.be/features/ [Accessed on 13th March 2017]

[18]http://openvpn.net [Accessed on 13th March 2017]

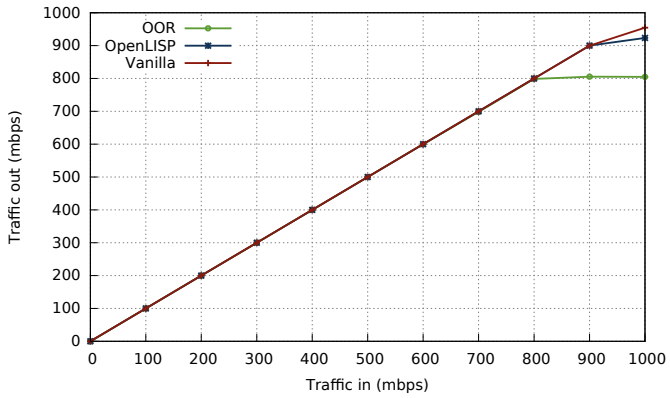

Fig. 3. **Throughput (Linux)**

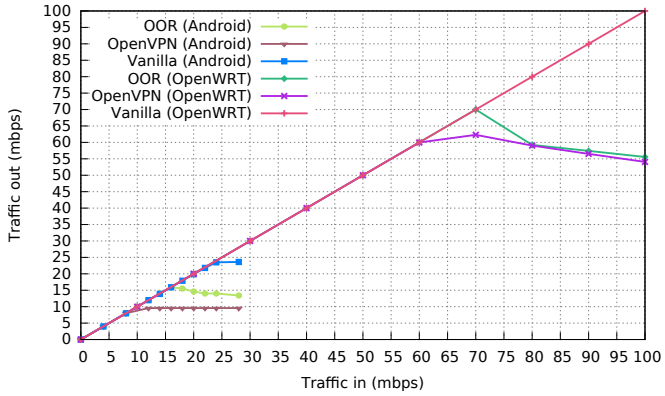

Fig. 4. **Throughput (Android and OpenWrt)**

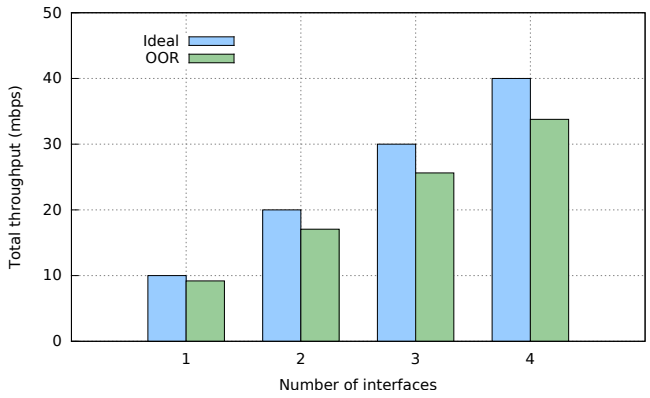

Fig. 5. **Multihoming performance**

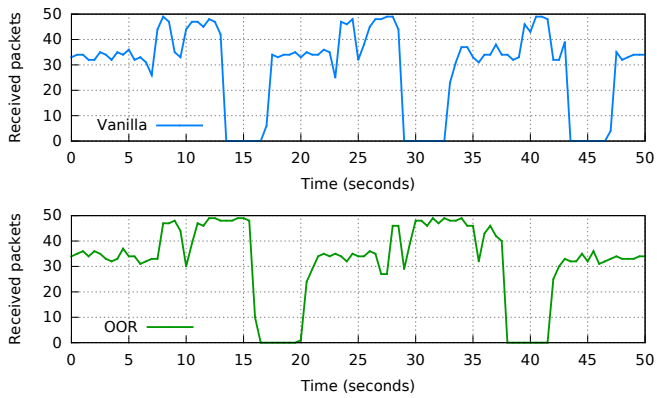

Fig. 6. **Handover time**

(WNDR3800, OpenWrt 12.09) and we generate traffic with *nuttcp*. As shown in Fig. 4 OOR outperforms OpenVPN in both cases. Both OOR and OpenVPN show a slight decrease in performance under high loads.

B. Multihoming

As described in section IV-B, OOR supports multiple data interfaces at the same time. In order to test the performance of OOR in this scenario, we run OOR in a virtual machine connected to 4 different interfaces (10Mbps of link capacity per interface). We generate 100 flows using *iperf* and we measure the average throughput as a function of the number of active interfaces. As Fig. 5 shows, OOR dynamically takes advantage of the available interfaces resulting in efficient multihoming. The overall throughput is limited by the overhead introduced by the different encapsulation headers.

C. Horizontal Handover Latency

We also focus on the handover latency across technologies on the Android implementation. With OOR running on our Nexus 7 tablet we manually force horizontal handovers (WiFi and 3G). At the same time, the tablet is generating a high ratio of ICMP packets (50 pkts/s) towards a remote host. The handover latency is measured as the time when the host is not receiving packets. Fig. 6 shows the number of packets received per second over a series of handover events with and without OOR. As shown in the plot, there are not noticeable differences between the scenario with or without OOR. This is due to the handover bottleneck being on the underlying hardware operations, out of OOR control. OOR reacts as fast as the kernel does to physical interfaces changes and it introduces negligible signaling latency. Indeed, the average WiFi to 3G handover latency measured over 31 tests is 4.16s and 3.98s with and without OOR respectively. From 3G to WiFi the handover latency does not impact the data-path since the Android OS does not turn off the 3G interface until there is WiFi connectivity. Nevertheless, it should be possible for OOR to improve the handover times shown in Fig. 6 via buffering or low level hardware management. However, at the time of this writing, that remains as future work.

IX. Conclusions

The OOR project offers a mature solution for LISP-based programmable overlays. Its modular user-space approach provides an extensible and flexible LISP implementation while keeping low complexity and easy deployment.

Additionally, experimental evaluation shows remarkable performance of the OOR control-plane in terms of processing and handover latency, resulting in unnoticeable overhead on the system. OOR's data-plane presents comparable performance to similar software solutions (e.g., OpenVPN). Such results show that OOR, although taking a full user-space approach, is suitable for production in edge and home environments.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their useful feedback and constructive comments. We also thank Shakib Ahmed for his help proofreading the manuscript.

This work has been partially supported by a Cisco research grant, by the Spanish Ministry of Education under scholarships FPU2012/01137 and AP2009-3790, 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] H. Eriksson, "Mbone: The multicast backbone," *Communications of the ACM*, vol. 37, no. 8, 1994, pp. 54-61.

[2] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan, "Best-path vs. multi-path overlay routing," In *Proc. of the 3rd ACM SIGCOMM Conf. on Internet measurement*, 2003, pp. 91-100.

[3] B. Y. Zhao *et al.*, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE JSAC*, vol. 22, no. 1, 2004, pp. 41-53.

[4] E. K. Lua *et al.*, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, 2005, pp. 72-93.

[5] A. Rodriguez-Natal *et al.*, "LISP: a southbound SDN protocol?," *IEEE Commun. Mag.*, vol. 53, no. 7, 2015, pp. 201-207.

[6] A. Rodriguez-Natal *et al.*, "Global State, Local Decisions: Decentralized NFV for ISPs via Enhanced SDN," *IEEE Commun. Mag.*, April 2017 (to appear).

[7] D. Farinacci *et al.*, "Locator/ID Separation Protocol (LISP)," IETF RFC 6830, Feb. 2013; https://tools.ietf.org/html/rfc6830 [Accessed on 13th March 2017]

[8] D. Saucez *et al.*, "Designing a deployable internet: the locator/identifier separation protocol," *IEEE Internet Comput.*, vol. 16, no. 6, 2012, pp. 14-21.

[9] A. Rodriguez-Natal *et al.*, "LISP-MN: mobile networking through LISP," *Springer Wireless personal communications*, vol. 70, no. 1, 2013, pp. 253-266.

[10] Y. Han *et al.*, "Design and implementation of LISP controller in ONOS," *NetSoft Conf. and Workshops (NetSoft)*, IEEE, 2016, pp. 417-422.

[11] T. Balan, D. Robu, and F. Sandu, "LISP Optimisation of Mobile Data Streaming in Connected Societies," *Mobile Information Systems*, vol. 2016, Article ID 9597579, 2016.

[12] F. Coras *et al.*, "An Analytical Model for Loc/ID Mappings Caches," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, 2016, pp. 506-516.

[13] S. K. Fayazbakhsh *et al.*, "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags," *11th USENIX Symp. on Networked Systems Design and Implementation (NSDI '14)*, vol. 14, 2014, pp. 533-546.

[14] D. C. Phung *et al.*, "The OpenLISP control-plane architecture," *IEEE Network*, vol. 28, no. 2, 2014, pp. 34-40.

[15] A. Stockmayer, M. Schmidt, and M. Menth, "jLISP: An Open, Modular, and Extensible Java-Based LISP Implementation," *28th Int. Teletraffic Congr. (ITC 28), 2016*, vol. 1, 2016, pp. 205-208.

Alberto Rodriguez-Natal received a BSc (2010) in Computer Science from the University of Leon (Spain) and a MSc (2012) and PhD (2016) from the Technical University of Catalonia (Spain). He has also been a visiting researcher (2014) at the National Institute of Informatics (Japan). He joined Cisco in 2016 where he continues his research on new network architectures with special focus on Software-Defined Networking and programmable overlay networks.

Jordi Paillisse received his degree in Telecommunications Engineering (2013) from the Technical University of Catalonia (Spain). He has been a visiting student in Ecole Polytechnique Federale de Lausanne (Switzerland). He is currently a PhD candidate in Computer Architecture at the Technical University of Catalonia. His main research interests are future Internet architectures and Software-Defined Networking.

Florin Coras is a Software Engineer in the Chief Technology and Architecture Office at Cisco Systems. He joined Cisco in 2015 and has since focused on research and development of SDN technologies, network virtualization and overlays. He has authored several academic papers and IETF documents that concern LISP protocol design and contributed to LISP implementations in OOR, VPP and ODL. Prior to joining Cisco, Florin received a PhD (2015) and a MSc (2011) degree in Computer Science from Universitat Politecnica de Catalunya (UPC), Barcelona, Spain and a Diplom Degree (2009) in Telecommunications Engineering from the Technical University of Cluj-Napoca.

Albert Lopez-Bresco received a BSc (2005) and a MSc (2007) in Telecommunications Engineering from the Technical University of Catalonia (Spain) where he is working as a Software Engineer. He is a core developer and principal maintainer of the Open Overlay Router project. He is also the main administrator for the LISP Beta Network (lisp4.net), a world-wide LISP pilot deployment.

Lorand Jakab is an open source enthusiast working on network virtualization solutions at Cisco. He contributed to almost all open source software related to LISP, and is a committer on the OOR and OpenDaylight LISP Flow Mapping projects. He participated in the LISP beta network large scale experiment, and co-authored an RFC an deployment considerations. He holds a Ph.D. from the Technical University of Catalonia.

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 Tecnologic de Telecomunicacions de Catalunya (CTTC) where he participated in multiple R&D projects. His current research interests are on SDN and network virtualization solutions.

Preethi Natarajan is a Technical Lead at the Chief Technology and Architecture Office at Cisco. Her research interests include SDN, network analytics, AQM technologies to tackle bufferbloat and QoS. Recently she has been working on integrating advanced analytics into Cisco's network management solutions. She received her PhD from University of Delaware in 2009.

Vina Ermagan is a Sr. Technical Leader 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 network virtualization and SDN technologies ever since. She has initiated projects to implement LISP in Open vSwitch (OVS), OpenDaylight, and FD.io . Vina received her MSc in Computer Science from UC San Diego in 2008, and her BSc in Computer Engineering from Sharif University of Technology.

David Meyer is currently Chief Scientist and Fellow at Brocade Communications. Previously he has been CTO (Chief Technology Officer) at Brocade for 3 years. Prior to join Brocade he was Distinguished Engineer at Cisco for 12 years where, among other contributions, he co-authored the Locator/ID Separation Protocol (LISP). He has also been appointed as chair of the OpenDaylight Technical Steering Committee (TSC) and as co-chair of the Software-Defined Networking (SDN) Research Group at the Internet Engineering Task Force (IETF). For the last 20 years he has been a senior research scientist at the Department of Computer Science at the University of Oregon. He has 35 Internet standards (RFCs), many papers in academic journals and conferences and several patents.

Dino Farinacci is the founder of lispers.net, a non-profit engineering consulting company dedicated to the advancement of the Internet. He received a BSc in Engineering from the Ohio State University and he has 35 years of experience in computer networks architecture and computer networks design. Prior to founding lispers.net he worked for Cisco Systems for 16 years, where he achieved the status of Cisco Fellow in 1997. For more than 25 years, he has also been serving as senior member of the IETF. He has over 45 patents in the networking field. His latest major contribution to the industry is the Locator/Identity Separation Protocol (LISP), a protocol to deploy and operate overlay networks.

Fabio Maino is a Distinguished Engineer at Cisco Systems, in the Chief of Technology and Architecture Office, where he leads a team that focuses on driving innovation on Network Virtualization and SDN. He has about 50 patents issued or filed with the US PTO, and has contributed to various standardization bodies including IEEE, IETF, and INCITS. Fabio has a Ph.D. in Computer and Network Security and a M.S. ("Laurea") in Electronic Engineering from Politecnico di Torino, Italy.

Dr. Albert Cabellos is an associate professor at Universitat Politcnica de Cataulnya (Department of Computer Architecture), he has been visiting professor at Cisco Systems (Santa Jose, US), KTH (Kista, Stockholm), Agilent Technologies (Edinburgh, UK), Massachusetts Institute of Technology (Cambridge, USA) and UC Berkeley (Berkeley, USA). He has participated in several research projects funded by companies (Cisco, Intel, Samsung, etc) as well as publicly funded (FP7, H2020, NSF and national funding agency). He is also a co-founder of the Open Overlay Router (http://openoverlayrouter.org) open-source project.