P4-LISP: A P4-Based High-Performance Router for the Locator/Identifier Separation Protocol

Benjamin Steinert*†, Marco Häberle*, Jan-Oliver Nick*, Dino Farinacci‡, Michael Menth*
* University of Tuebingen, Chair of Communication Networks, Tuebingen, Germany
† University of Tuebingen, Zentrum für Datenverarbeitung, Tuebingen, Germany
‡ lispers.net, California, USA
Email: {benjamin.steinert,marco.haeberle,menth}@uni-tuebingen.de,
jan-oliver.nick@student.uni-tuebingen.de, farinacci@gmail.com

***Abstract*—The networking paradigm locator/identifier split decouples locating and identifying functionality of addresses. Thereby it improves multi-homing, fail-over, mobility, traffic engineering over the Internet, and routing scalability.**

The Locator/Identifier Separation Protocol (LISP) is a prominent incarnation of that paradigm which recently became an Internet standard. However, existing LISP implementations are either proprietary or have limited performance, which makes their deployment difficult in high-speed networks. Programming Protocol-independent Packet Processors (P4) is a programming language that facilitates the implementation of custom data plane processing on high-performance switches with line rates of up to 400 Gbit/s.

In this work, we present P4-LISP, an open-source P4-based proof of concept implementation of a high-performance LISP router. It supports all relevant features such as ITR, ETR, RTR, P-ITR, P-ETR, NAT-traversal, LISP-NAT, and mobile nodes. As control plane, the open-source implementation lispers.net has been integrated on the switch. Security features are added to protect the control plane from being overloaded by the high-performance data plane. The paper describes the architecture of P4-LISP in detail and extensively evaluates performance, functionality, controller performance, and overload protection.

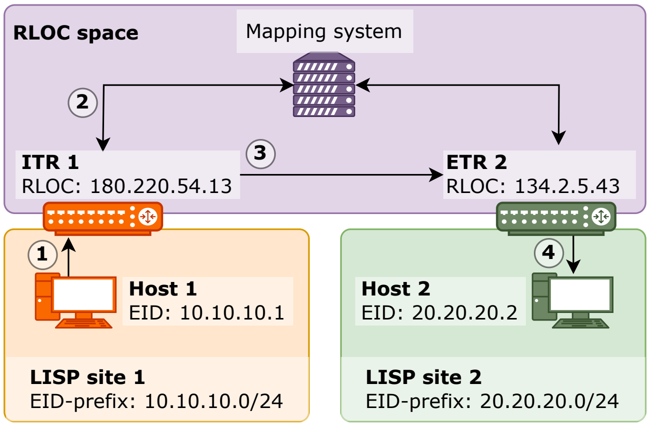

Fig. 1. LISP hosts are addressed with EIDs which are not globally routable. In contrast, xTRs have globally routable addresses, the RLOCs. For inter-domain communication, packets from within a LISP domain are encapsulated by ITRs with routable RLOCs, effectively forming an overlay over the Internet. The destination ETR decapsulates the packets and forwards them into the destination LISP domain.

I. Introduction

In today's Internet, IP addresses both identify a host and locate it in the Internet for steering packets to their destinations. The locator/identifier split (Loc/ID split) paradigm decouples both functions [1], which improves multi-homing, fail-over, mobility, traffic engineering (TE) over the Internet, and routing scalability. This is essentially achieved through overlay networking.

The Locator/Identifier Separation Protocol (LISP) [2] implements Loc/ID split for use in today's Internet and has become IETF standards track in October 2022 [3]. LISP operation is visualized in Figure 1. LISP defines LISP domains where domain hosts are addressed with Endpoint Identifiers (EIDs). These EIDs are public IP addresses which are not routable in the Internet as their prefix has not been announced through BGP. This improves global routing scalability as it saves space in the BGP routing tables. However, IP packets addressed to an EID can be forwarded within their local LISP domain, but not via the Internet. For inter-domain communication, border routers of LISP domains serve as tunnel routers, e.g., to other LISP domains. Such a tunnel router has a public IP address from its Internet service provider which is routable on the Internet and is called Routing Locator (RLOC). Thus, IP addresses used for LISP are subdivided into an EID and RLOC space. A tunnel router encapsulating packets from within a LISP domain with its RLOC is called Ingress Tunnel Router (ITR) while a tunnel router decapsulating traffic for a LISP domain is called Egress Tunnel Router (ETR). A router implementing both ITR and ETR functionality is called xTR.

Figure 1 illustrates how hosts from different LISP domains communicate with each other. Host 1 in LISP site 1 sends a packet to Host 2 in LISP site 2 using its EID as destination address (1). When the packet is received by ITR 1, it requests the RLOC of ETR 2 from the LISP mapping system (2). The LISP mapping system is a distributed database which holds EID-prefix-to-RLOC mappings for all EID-prefixes. Then, ITR 1 encapsulates the packet with that RLOC as destination address (3). Based on this RLOC, the packet is forwarded to ETR 2 over the Internet. ETR 2 decapsulates the packet and forwards it to Host 2 in LISP site 2 (4).

This example describes only the very principle of LISP. EID-prefix-to-RLOC mappings are stored by ITRs in a so-

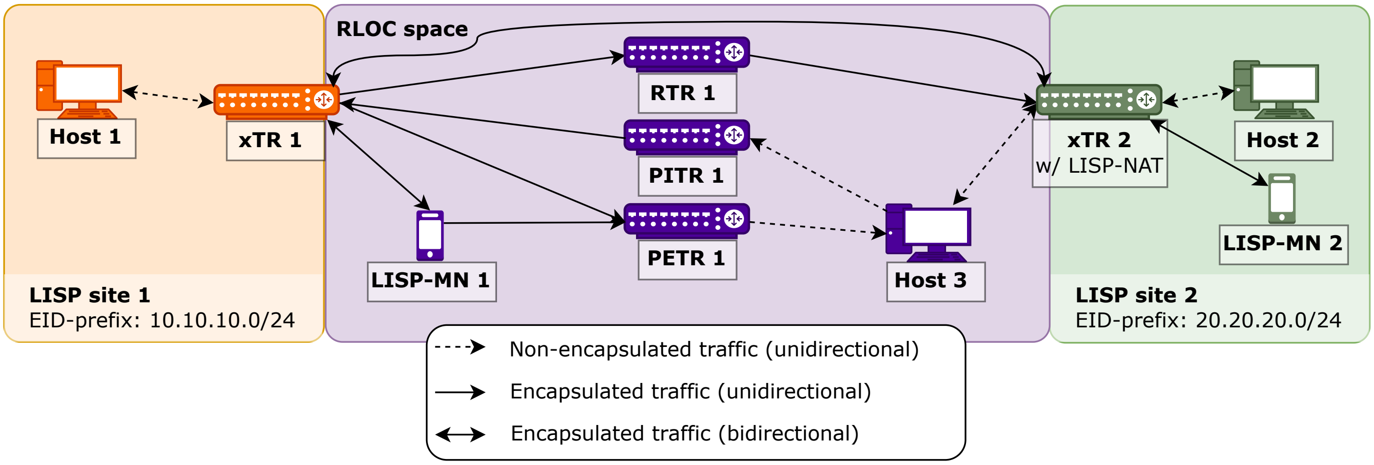
RLOC space
LISP site 1
EID-prefix: 10.10.10.0/24
Host 1
xTR 1
RTR 1
PITR 1
PETR 1
LISP-MN 1
Host 3
xTR 2
w/ LISP-NAT
Host 2
LISP-MN 2
LISP site 2
EID-prefix: 20.20.20.0/24
Non-encapsulated traffic (unidirectional)
Encapsulated traffic (unidirectional)
Encapsulated traffic (bidirectional)

Fig. 2. LISP topology containing all LISP router types, including xTR, RTR, PITR, PETR, and LISP-MN. All relevant LISP functionalities are shown. First, Tunneling between xTR 1 and xTR 2. Second, interworking between Host 1, Host 2, and Host 3. Third, LISP-MNs, and fourth, the usage of an RTR for NAT traversal or TE.

called Map-Cache such that lookups are rarely needed. When an ITR does not have an appropriate mapping for an EID, the ITR may either store or drop the packet while waiting for the mapping, which may cause packet loss in rare cases. In addition, LISP defines mechanisms for interoperability with hosts in non-LISP domains (LISP-NAT, PITR, and PETR), for mobile LISP hosts, for communication of LISP hosts behind a Network Address Translation (NAT) (NAT traversal), and for traffic engineering (TE) over the Internet (RTR).

Programming Protocol-independent Packet Processors (P4) [4] is a programming language for programming the data plane of forwarding devices. Some vendors offer it as an API to their customers such that they can realize own hardware innovations. Amongst others, data plane programming with P4 is available for Intel's high-performance ASIC Tofino which supports line rates of up to 400 Gb/s [5].

While different implementations of LISP exist, both in hardware and in software, they are either proprietary or have limited performance. This makes deployment of LISP difficult especially in high-speed networks.

In this paper, we present P4-LISP, an open-source P4-based data plane implementation of LISP that can be deployed on high-performance hardware like Tofino. P4-LISP supports all LISP router types (xTR, PxTR, RTR), LISP-NAT, mobile nodes, and NAT traversal. The data plane of the forwarding device is controlled by a local controller written in Python which leverages the open-source implementation lispers.net [6] as LISP control plane. P4-LISP includes elementary security measures to protect the local controller and the control plane against overload from the data plane, which may happen due to Denial of Service (DoS) attacks. These security measures include, e.g., filtering of packets based on outer and inner headers such as IP addresses.

The rest of the paper is structured as follows. Section II provides more background on LISP and P4 and Section III presents related work. The architecture and proof of concept (PoC) implementation of P4-LISP are described in Section IV. The performance of the PoC is evaluated in detail in Section V. Finally, we draw conclusions in Section VI.

II. Technical Background

In this section, we explain relevant LISP functionality in detail, such as tunneling, interworking, NAT traversal, and mobile nodes. Additionally, we briefly describe lispers.net which implements the LISP control plane, and give a short overview of P4.

A. *LISP Tunneling*

Communication between two LISP sites uses tunneling, effectively forming an overlay over the Internet, which works as follows. As presented in Section I, the ITR of a LISP domain performs a so-called map&encaps operation for outgoing traffic while the ETR performs a so-called decaps operation for incoming traffic. An ITR has a so-called Map-Cache to store EID-to-RLOC mappings that are retrieved by the mapping system. LISP encapsulation includes a UDP header, a LISP header for LISP-specific signaling information, and an outer IP header. In the UDP header, the source port is chosen by the ITR while the destination port is 4341. In the outer IP header, the routable RLOC addresses of the ITR and ETR are set as source and destination addresses. An example of inter-domain communication is displayed in Figure 2, between xTR 1 and xTR 2.

B. *LISP Interworking*

Interworking between LISP sites and non-LISP sites comes with two general challenges. First, LISP EIDs are not globally routable, therefore they cannot be directly reached on the Internet. Second, it may be the case that a non-routable EID cannot be used as source address, e.g., because it is blocked by the access network due to Reverse Path Forwarding (RPF) [7].

To overcome these challenges, RFC 6832 [8] suggests three different mechanisms to enable inter-networking between

LISP and non-LISP sites, for both IPv4 and IPv6. These mechanisms include LISP-NAT as well as Proxy Ingress Tunnel Routers (PITRs) and Proxy Egress Tunnel Routers (PETRs).

LISP-NAT is implemented on ITRs and treats IP packets with an EID as source IP address and an external, routable destination IP address just like a conventional NAT. That means, it rewrites the EID source address into a routable IP address using Network Address and Port Translation (NAPT), and it performs the inverse operation on the destination address for reverse traffic. Thereby, LISP nodes communicate with non-LISP nodes without LISP encapsulation. Figure 2 shows an example for LISP-NAT-enabled communication between xTR 2 and Host 3. LISP-NAT has two disadvantages: it requires an internal NAT table and works only for flows initiated from within LISP domains.

PITRs and PETRs provide an alternative interworking mechanism that also supports flows originating from a non-LISP site to a LISP site.

A PITR is an ITR that makes large EID-prefixes routable on the Internet by announcing them in BGP. As a consequence, traffic destined for such EIDs is attracted by that PITR on the Internet. The PITR performs map&encaps on these packets so that they reach their final destination. Figure 2 shows an example where Host 3 sends packets to xTR 1 via PITR 1. A drawback of this method is that it requires BGP announcements and leads to path stretch due to triangular routing.

A PETR is an ETR that helps flows from within a LISP site to a non-LISP site to leave their domain although the source EID is not routable on the Internet. To that end, such traffic is tunneled from the ITR of the LISP site to a configured PETR which decapsulates the traffic. From there, it reaches its destination. In Figure 2, communication from xTR 1 to Host 3 uses PETR 1. Also, a PETR may lead to path stretch.

The proxy mechanism also enables communication via IP protocols that are not supported by a LISP site. As an example, a LISP ITR only has IPv4 Internet connectivity but wants to reach an IPv6-only host on the Internet. By the use of a dual-stacked PETR, i.e., with IPv4 and IPv6 connectivity, IPv6 packets can be encapsulated by the ITR with an IPv4 header to be transported over the IPv4-only access network to the PETR. For the reverse direction, a dual-stacked PITR receives IPv6 packets and encapsulates them with an IPv4 header. This mechanism is called mixed-protocol encapsulation.

C. LISP Mobile Node

LISP supports host mobility. The concept of a LISP Mobile Node (LISP-MN) facilitates that a mobile host communicates with a stable EID which is independent of its location [9]. A LISP-MN constitutes its own LISP domain and implements xTR functionality. When a LISP-MN roams into a non-LISP site, it obtains a routable IP address which it utilizes as RLOC, e.g., LISP-MN 1 in Figure 2. When the RLOC set of a LISP-MN changes, this needs to be signaled via the LISP control plane.

When the IP address of a mobile non-LISP node changes, connections like TCP are terminated as they utilize the IP address as part of the connection identifier. With LISP-MN, such connections survive roaming events as the EID is used for connection identification instead of the ephemeral IP address. For communication with non-local, non-LISP destinations, LISP-MNs utilize a PETR.

When a LISP-MN roams into a LISP site, i.e., behind another xTR, it may receive one of the LISP domain's EIDs as RLOC, e.g., LISP-MN 2 in Figure 2. The ITR of the LISP site receives LISP-encapsulated packets by the LISP-MN with an EID as source address. In this scenario, communication with the LISP-MN may result in double encapsulation of packets.

D. RTR: TE over the Internet and LISP NAT Traversal

A Re-encapsulating Tunnel Router (RTR) is a special LISP router type that accepts LISP-encapsulated packets, decapsulates, and re-encapsulates them for delivery to another RLOC. This mechanism enables TE over the Internet and NAT traversal.

For TE over the Internet, an ITR can relay packets via one or more RTRs, to enforce a pre-defined path towards a certain EID. In the mapping system, specific RTR paths can be stored for EID-prefixes via the so-called Explicit Locator Path (ELP) encoding defined in RFC 8060 [10]. If such a mapping is received by an ITR, it encapsulates packets towards the respective EID-prefix with the first RTR locator in the list. This RTR then decapsulates the packet and again encapsulates it with the next RTR locator in the list. This continues until the packets are received by the destination RLOC.

In many cases, e.g., for LISP-MNs, an xTR is deployed behind a NAT. Then, the RLOC of the xTR is not globally routable and only reachable via a NAT's public address. Additionally, the RLOC is not reachable until a translation state has been established by the NAT device. A method of enabling communication to/from xTRs behind NATs includes the use of RTRs as anchor points [11]–[13].

E. LISP Control Plane lispers.net

Lispers.net [6] is an open-source Python implementation of a LISP control plane, data plane, and mapping system [14]. The control plane implementation conforms to RFC 9301 [15] and allows to use an external data plane. The communication between the lispers.net control plane and an external data plane is realized using IPC messages via sockets. IPC messages are transmitted in JSON format. According to RFC 9301, different message types are used for signaling the forwarding information between xTRs and the mapping system. Map-Request and Map-Reply messages are used for requesting and delivering mapping information. Map-Register and Map-Notify messages are used for registering and updating EID-to-RLOC mappings.

F. P4 Fundamentals

P4 is a domain-specific programming language for the implementation of data plane logic in a programmable switch

[4]. A P4 program can be compiled for various so-called P4 targets. These P4 targets may be software switches such as BMv2 [16], or hardware switches, e.g., based on the Intel Tofino ASIC [5]. P4 also supports the use of target-specific functionalities that can be called via so-called *extern* functions, which makes the code target-specific.

A P4 program typically consists of the following key components. A parser includes the definition of headers, a match-action pipeline consisting of Match-Action Tables (MATs), and an overall processing logic. The parser is represented by a finite state machine and can parse all headers that have been defined in the P4 program, which facilitates processing of custom headers. The MATs can be applied in a custom order and are usually provisioned by a P4 controller. The P4 controller communicates with the data plane using a runtime interface. Stateful packet processing is possible using so-called registers for storing information beyond the processing of a single packet. More information on P4 is provided in an extensive survey by Hauser et al. [17].

III. Related Work

Multiple protocols implement the locator/identifier split, e.g., the Identifier-Locator Network Protocol (ILNP) [18], the Host Identity Protocol (HIP) [19], Shim6 [20], or the Locator/Identifier Separation Protocol (LISP) [3] which we will focus on in this work. LISP has been used in different use cases that benefit from multi-homing, mobility, and TE capabilities. A selection of these works is presented in the following, together with an overview of existing LISP implementations.

A. LISP Use Cases

LISP has been used in various works to increase resilience and achieve high availability by leveraging LISP multi-homing capabilities [21], [22]. By combining LISP with Multipath TCP (MPTCP), Coudron et al. were able to use different network paths simultaneously to increase file transfer speeds [23]. Additionally, this combination allows mobile devices to use multiple wireless interfaces at the same time.

Since the EID of a host or service does not depend on its location, LISP allows to identify and reach virtualized services via the same addresses independently of their deployment location. This mobility support is especially useful for data center operations since it allows seamless Virtual Machine (VM) mobility together with uninterrupted reachability [24].

Similarly, LISP can improve multi-link communication, e.g, for airplanes with seamless handover between different connection points [25]–[27], or improve mobility management in software-defined wireless networks [28].

LISP can also be used for traffic engineering over the Internet without complex Border Gateway Protocol (BGP) configurations [29]. Farinacci et al. defined Explicit Locator Paths to use RTRs for TE in intra-domain and inter-domain scenarios [30]. Due to its TE capabilities, LISP has been used for service function chaining [21] or for attack mitigation that keeps an attacker oblivious of the mitigation effort [31]. LISP overlays can work on a number of different underlays, e.g., over satellite networks [32]. Today, LISP is mostly used for secure cloud and VPN access, or SD-WAN.

B. LISP Implementations

Several implementations of LISP are available for the data plane as well as for the control plane. Most of them are either proprietary or have limited performance.

Software implementations of LISP include OpenLISP, Pylisp, jLISP, or Open Overlay Router (OOR). OpenLISP [33] is an early open-source implementation of LISP for FreeBSD. OpenLISP's data plane runs in kernel space and includes LISP-Cache, LISP-Database, and encapsulation and decapsulation capabilities. PyLISP [34] is a Python implementation of LISP that provides basic data plane and control plane functions for an xTR. jLISP [35] is written in Java. It is flexible and can run on various platforms, but has limited performance. Open Overlay Router (OOR) [36], which started as the LISPmob.org project, is a flexible and modular open-source software implementation of an xTR, LISP-MN, mapping system, Delegated Database Tree node, and RTR. Additionally, LISP is implemented and used in several Software-defined Networking (SDN) related projects, e.g., ONOS, OpenDaylight, or Open vSwitch. The SDN controllers Open Network Operating System (ONOS) [37] and OpenDaylight [38] support the use of LISP as a southbound SDN protocol [39]. Open vSwitch [40] is an open-source implementation of a distributed virtual software switch that supports LISP as layer 3 tunneling protocol since 2013.

For LISP, multiple mapping systems have been presented [41]. Among them, FIRMS forwards packets when ITRs have no appropriate mapping in their map-cache to avoid packet loss [42].

Proprietary hardware-based implementations are provided by, e.g., Cisco, AVM, or LANCOM. The implementation from Cisco is used within a wide range of their router and switch operating systems (IOS, IOS XR, IOS XE, and NX-OS). The solution provides functionalities of xTR, mapping system, PxTR, and RTR [43]. AVM's FRITZ!Box home routers support xTR functionality since Fritz!OS 6.00 [44]. LANCOM Systems supports xTR functionality in their router operating system since version 10.20 [45].

IV. Architecture and Implementation of P4-LISP

In the following, we give an overview of the architecture of P4-LISP, followed by a description of the data plane and control plane operation, including details of the PoC implementation that is published under an open-source license on GitHub [46].

A. Architecture

The P4-LISP router architecture, visualized in Figure 3, consists of the P4 data plane implementation, a local P4 controller, and a LISP control plane.

The data plane of the LISP xTR stores EID-to-RLOC mappings in a LISP Map-Cache, performs encapsulations, decapsulations, re-encapsulations, filtering, and IP processing.

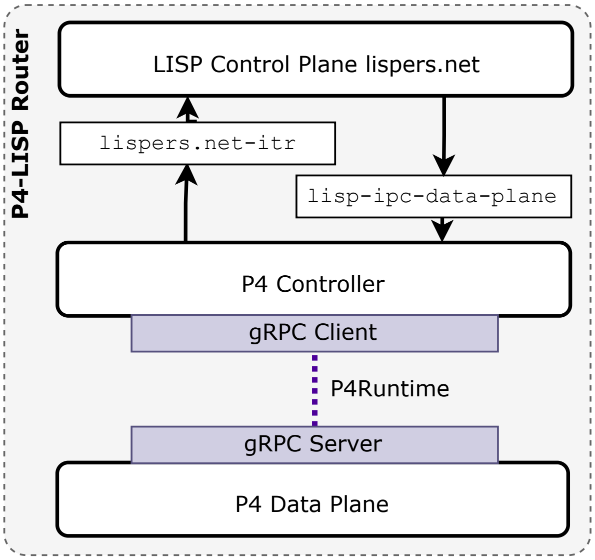

Fig. 3. Architecture of the P4-LISP router. The communication between the local P4 controller and the data plane is established using the gRPC-based P4Runtime API. Messages between the P4 controller and the control plane are transmitted via `AF_UNIX/SOCK-DGRAM` sockets.

For interworking or NAT traversal mechanisms, header fields like source or destination address, or port numbers are altered. For the support of mobile nodes, double encapsulation is supported.

We use a high-performance Intel Tofino ASIC integrated within an Edgecore Wedge 100BF-32X switch [47] to deploy the P4 implementation of the data plane.

The P4 controller inserts and deletes Map-Cache entries on the data plane, and communicates with the LISP control plane. It is implemented using Python and uses the P4Runtime [48] for communication with the P4 data plane implementation. The open-source LISP control plane lispers.net [6] is utilized as mapping system or for communication with a remote mapping system that resolves LISP Map-Requests.

B. P4 Data Plane Layout

The data plane of the P4-LISP router is implemented as a P4 match-action pipeline and is shown as a simplified flowchart in Figure 4. Each packet entering the switch, LISP-encapsulated or not, is processed by this pipeline in line rate, including standard IP ingress- and egress-processing. The pipeline consists of match-action tables that are provisioned by the controller via the P4Runtime. Depending on the configuration of the router, the tables will have different entries with varying match criteria and action sets.

Packets entering the pipeline are first processed by the *decapsulation table*. For packets with destination port 4341, the packet is decapsulated if the destination address matches one of the router's RLOCs. In this case, the outer IP header, the UDP header, and the LISP header are removed, leaving only the inner IP header. Otherwise, no action is performed.

The following two tables, i.e., the *allowed prefix table* and the *valid destination table*, can be used to filter packets based on source or destination address of the inner or outer IP header. The *encapsulation table* represents the LISP Map-Cache. It holds the Map-Cache entries whose number may be in an order of magnitude of around 100 K.

If the router is configured to act as an xTR and is located behind a NAT, the encapsulation table replaces the UDP source port with a random static value that is provided by the control plane. This source port will be translated by the NAT to a value that is known by the respective RTR.

If the router is configured to act as an RTR, packets will be subsequently decapsulated and encapsulated in the same pipeline. In this case, the UDP source port is set to 4342 in the encapsulation table, and the UDP destination port is set to a specific UDP port that matches the translation state in the NAT.

If the router is configured to act as an xTR and if LISP-NAT is enabled for communication to non-LISP hosts, the *LISP-NAT tables* act as NAT translation tables. They are used to replace the private EID with a translated globally routable EID that is assigned by the controller. For outbound packets destined to a non-LISP enabled host, the source address is adapted in table *LISP-NAT (outbound)*. In the reverse direction, the globally routable destination address is changed to the translated private EID in the *LISP-NAT (inbound) table*. The P4 controller provisions the respective table entries to realize the translation between the globally routable EIDs and the private EIDs.

In case of missing forwarding information, e.g., when a cache miss happens, packets are sent to the local P4 controller using the *IPv4 table*. Subsequently, the P4 controller issues Map-Requests for the respective destination by leveraging the lispers.net control plane, and writes requested Map-Cache entries back to the data plane. Finally, the forwarding table contains information about the respective next hop and the packet is forwarded with the respective Ethernet header.

Note that the IP forwarding table is directly implemented in the same data plane pipeline as LISP router operations. This allows non-LISP packets to be processed by the router in the same pipeline with no added latency compared to LISP packets, different from most existing implementations.

Also, note that an xTR is typically deployed behind a NAT and therefore has a 0.0.0.0/0 Map-Cache entry towards an RTR installed. If the destination is not an EID, the packets may be forwarded without encapsulation on a default route in the underlay. So only in very few cases the packet is forwarded to the controller and a Map-Request is sent.

C. P4 Data Plane Security Measures

Attackers can perform a DoS attack against the P4 controller of P4-LISP and the LISP control plane by sending lots of traffic with EIDs that are not yet in the Map-Cache. This triggers a message from the data plane via the P4 controller to the LISP control plane so that both may be overloaded. This is a general problem of LISP. However, P4-LISP implements measures to counteract such attacks and reduce load on the control planes.

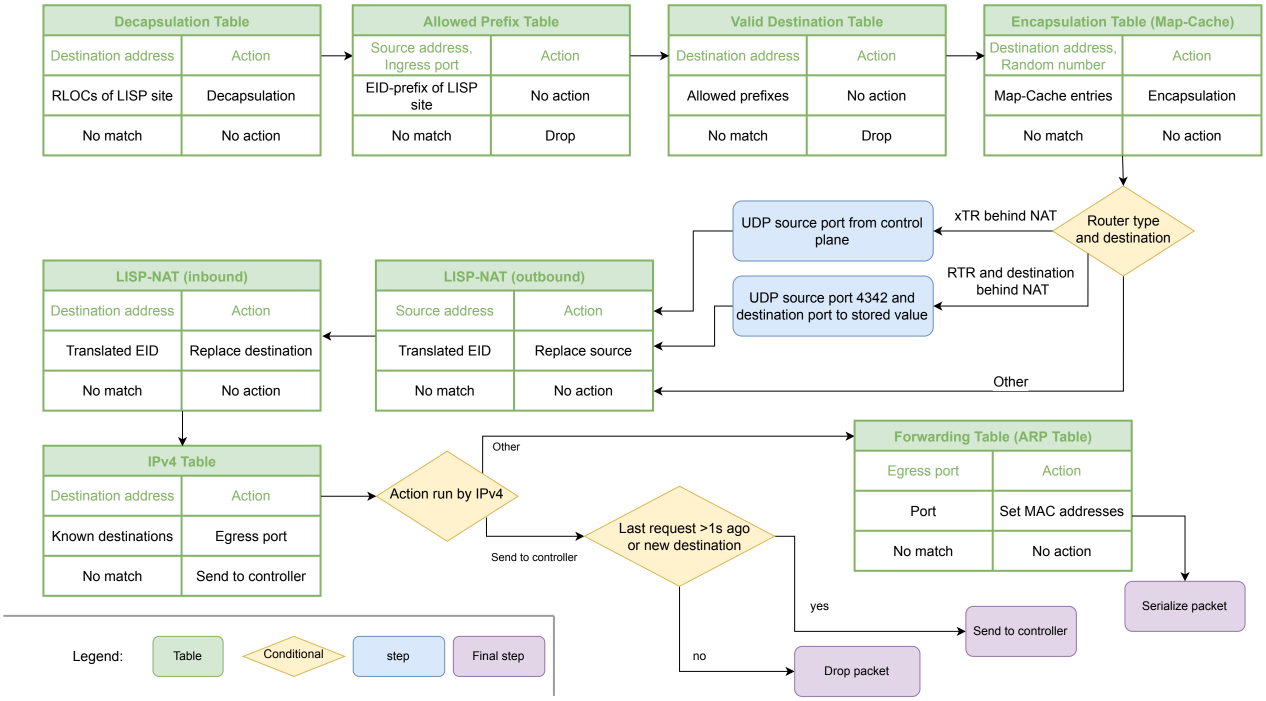

Fig. 4. Flowchart for the logic of P4-LISP's MAT pipeline.

P4-LISP supports white- and blacklisting specific outer or inner source and destination addresses by allowing or disallowing them with the *allowed prefix table* and the *valid destination table*. Thereby EID-prefixes of known communication partners may pass and EID-prefixes of known spammers that always use the same source addresses may be blocked. This is efficient with P4-LISP as filtering encapsulated packets based on outer or inner headers is performed at line rate.

Second, packets to the controller are rate-limited to one packet per second per destination EID. This mechanism is implemented in P4 using registers that can be used to read and store information on the data plane, and can be manipulated by the P4 controller as well. For each packet that is forwarded to the local controller, a register entry is created on the data plane. This register entry contains a hash of the destination EID and represents a pending Map-Request for this EID. Before forwarding a packet to the controller, the data plane verifies if a register entry for the hashed destination address exists. If yes, the packet is dropped instead of forwarded to the controller since there already is a pending Map-Request for the respective destination EID.

On arrival of a packet at the controller, it stores the destination EID together with a timestamp. Once the EID-to-RLOC mapping has been transmitted to the data plane, the controller deletes the respective register entry. At latest, after one second, the register entry on the data plane is deleted by the controller, independent of the state of the pending Map-Request. This effectively blocks the triggering of subsequent Map-Requests for the same destination. This rate-limiting ensures that the local P4 controller does not receive an unnecessarily high amount of packets, and the issuing of Map-Requests is also limited to one per second, as required by RFC 9301 [15].

D. lispers.net Control Plane

We integrate the open-source control plane lispers.net [6] as LISP control plane with the hardware-based P4-LISP data plane implementation. lispers.net offers a data plane API that allows using the control plane with different software or hardware forwarding implementations. For Map-Cache population and other control plane communication, it specifies an Inter Process Communication (IPC) interface for which a Unix socket is created by the control plane. The IPC messages sent to the socket follow a specific format found on the respective GitHub page [14] and are encoded in JSON format. For the socket, an `AF_UNIX/SOCK-DGRAM` socket called `lisp-ipc-data-plane` is used. The path name of the named socket resides in the lispers.net directory. The data plane code is responsible for creating the named socket in the above directory, such that the data plane and the lispers.net code can communicate.

E. P4 Controller

The P4-LISP data plane is provisioned and controlled by a local SDN controller that communicates with the data plane via the gRPC-based P4Runtime [48]. In this work, the local P4 controller is implemented using Python and runs directly on the x86_64 management platform of the Edgecore switch. Apart from controlling the data plane, the local P4 controller also connects to the control plane via a local socket. Packets can be forwarded from the data plane to the controller via the local CPU port of the Tofino ASIC. These packets are translated by the controller into IPC messages that are sent to the control plane. Additionally, it translates mapping information of replies from the control plane into MAT entries that are provisioned to the data plane.

V. Evaluation

In this section, P4-LISP is evaluated. We start with a description of the evaluation setup based on the presented prototype. Then, four different evaluations are performed. First, a large set of test scenarios is conducted to verify the functionality of the implementation with different configurations and in different topologies. Second, a bandwidth test is performed to verify that packets are processed at 100 Gbit/s line rate. Third, the end-to-end latency is analyzed including communication with the local P4 controller and the control plane. Last, a stress test examines the maximum load that the P4 controller and the control plane can handle.

A. Evaluation Setup

For the evaluations, we use VMs as senders and receivers, running on two VM hosts. Each VM host runs Proxmox Virtual Environment 6.4 and is configured with an Intel Xeon Gold 6134 CPU running at 3.2 GHz, 128 GB RAM, and two 100G Mellanox ConnectX-5 Network Interface Cards (NICs). The sender and receiver VMs run Ubuntu 20.04 and are configured with 4 virtual CPU cores, 16 GB RAM, and 1 ConnectX-5 NIC that is integrated using SR-IOV. No overbooking is performed on the VM hosts and the virtual CPUs are pinned to physical cores to minimize the influence of virtualization on the experimental results. All evaluations of the P4 program are performed on the Edgecore Wedge 100BF-32X with an Intel Tofino programmable ASIC using SDE version 9.7.0. The local P4 controller and the lispers.net control plane run directly on the x86_64 management platform of the Edgecore switch, equipped with an Intel Pentium D1517 CPU running at 2.2 GHz, and 8 GB RAM. Packets from the data plane to the controller are sent via the local CPU port of the Tofino ASIC, while the connection between the controller and the lispers.net control plane is realized using local sockets.

B. Functional Unit Tests

A large set of scenarios and use cases is evaluated to verify the functionality of the P4 program with different configurations and in different scenarios. The testbed setup is similar to the one in Figure 5. The P4 program was configured to act as xTR, PITR, PETR, and RTR, respectively. For each of these router types, numerous test scenarios have been evaluated which are listed in Table I. Amongst others, we tested both interworking mechanisms (PITR-based interworking and LISP-NAT-based interworking), as well as NAT traversal using RTR. Each of these unit tests was executed successfully, verifying the ability of the P4 program to act as LISP router in different scenarios and deployments.

TABLE I
Unit tests for different LISP router types.

Router	No.	Test description
xTR	1	Packet encapsulation
	2	Packet decapsulation
	3	Load balance to multiple ETRs
	4	Drop packet with invalid source address
	5	Drop packet towards disallowed destination
	6	NAT traversal
	7	LISP-MN to lisp node
	8	LISP-MN to non-LISP endpoint
	9	LISP-MN to LISP-MN (EID)
	10	LISP-MN to LISP-MN (RLOC)
	11	Node to LISP-MN in other LISP domain
	12	LISP-MN receives packet
	13	Native Forwarding without encapsulation
	14	LISP-NAT outbound traffic
	15	LISP-NAT inbound traffic
PITR	16	Encapsulate for non-LISP host
	17	Encapsulate from LISP-MN (RLOC)
	18	Encapsulate from non-LISP host to LISP-MN (EID)
	19	Drop packet with invalid source outside
PETR	20	Decapsulate packet with one encapsulation
	21	Decapsulate packet with two encapsulations
	22	Drop encapsulated packet with non-registered source address
	23	Drop twice encapsulated packet with non-registered source address
RTR	24	Send packet towards node behind NAT
	25	Send packet towards LISP-MN behind NAT
	26	Receive packet from node behind NAT for encapsulation
	27	Receive packet from node behind NAT for encapsulation to LISP-MN
	28	Receive packet from node behind NAT to forward without encapsulation
	29	Drop packet if neither source nor destination is known

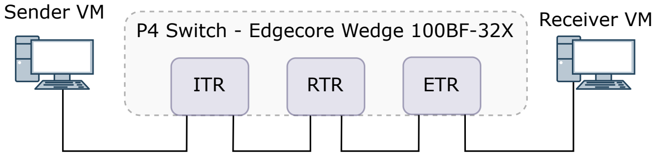

Fig. 5. Testbed setup for bandwidth evaluation. The sender VM sends TCP traffic toward the receiver VM via the P4 switch. The P4 switch acts as ITR, RTR, and ETR on different ports.

C. Forwarding Performance

For bandwidth evaluation, we use the testbed setup shown in Figure 5. On the sender VM, ten iperf3 processes are used to generate and send TCP traffic toward the P4 switch. Linux buffer sizes are set to 64 MB and TCP window size is set to 500 KB. On the P4 switch, the P4 program was modified to perform encapsulation, re-encapsulation, and decapsulation on incoming packets subsequently. On high-performance data plane devices, RTR functionality is particularly challenging at line rate since it requires both decapsulation and encapsulation. Respective Map-Cache entries have been provisioned to the data plane in advance. During this evaluation, no packet drops were observed and the observed TCP goodput was approx. 90 Gbit/s which is close to 100 Gbit/s L1 capacity. Thus, the setup is able to perform all tested operations at line rate, demonstrating that it is possible to perform decapsulation and re-encapsulation of the same packet at line rate on the P4 device.

D. Latency Analysis

We measure the latency that is introduced by the communication between the data plane, the local P4 controller, and the lispers.net control plane when a new flow is established. For this purpose, we consider that a host in a LISP site sends traffic to another host in a different LISP site.

Initially, we assume that the appropriate EID-to-RLOC mapping is not available in the Map-Cache of the P4-LISP router so that a Map-Request has to be issued to retrieve the mapping. This assumption causes a worst-case latency which happens only if the destination is an EID that has to be resolved by an xTR or an RTR. Usually, e.g., for an ITR behind a NAT that runs NAT-Traversal logic, a default map-cache is present such that packets will never experience the latency arising from communication to the controller.

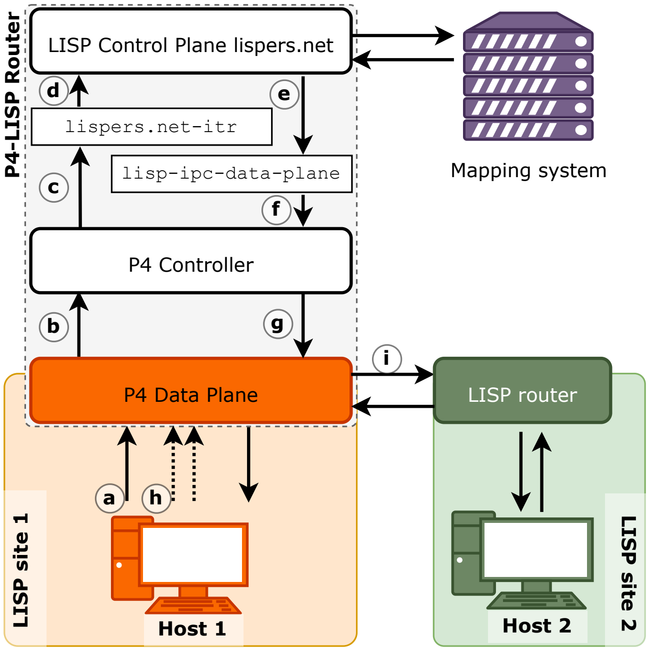

Fig. 6. Communication to a destination where no EID-to-RLOC mapping is present in the Map-Cache of a LISP router requires communication with the mapping system to retrieve the respective mapping. Involved components include the P4 data plane, the P4 controller, and the LISP control plane lispers.net.

Figure 6 illustrates the communication with the control plane when Host 1 starts sending packets to Host 2 and the EID-to-RLOC mapping for the EID of Host 2 is looked up before the xTR can forward the packets using map&encaps.

For the evaluation, 50 packets are sent per second from the source Host 1 to a local xTR of the LISP site using nping [49] (**a**). For each packet that is sent, a timestamp of the sending time is stored in a packet capture of the sending interface. There is no EID-to-RLOC mapping in the Map-Cache of the xTR, therefore the first packet is forwarded to the P4 controller (**b**). The P4 controller parses the packet, and sends a discovery message to the *lispers.net-itr* socket of the control plane (**c**). A timestamp is stored at the write time to the lispers.net-itr socket. The control plane receives the packet on the socket (**d**), sends a Map-Request to the mapping system, and receives a Map-Reply. The information in the Map-Reply is then sent to the *lisp-ipc-data-plane* socket (**e**).

TABLE II
MEAN AND STANDARD DEVIATION (SD) OF THE LATENCY CAUSED BY A CACHE-MISS IN THE P4-LISP DATA PLANE.

Steps	Mean (ms)	SD (ms)	Description of the steps
a-i	19.74	0.81	Worst-case latency: First packet sent by source until received by destination.
a-c	4.45	0.16	P4 controller delay for issuing discovery message.
d-e	12.61	0.72	Control plane delay.
f	0.80	0.10	P4 controller delay for processing control plane answer.
g-i	1.78	0.20	P4 controller delay for inserting table entry, until packet reaches destination.

The P4 controller receives the mapping information on the socket, stores a timestamp on reception, processes the answer (**f**), and adds a respective table entry to the Map-Cache table of the data plane (**g**). Another timestamp is stored when the gRPC message is sent to the P4 switch. Now, the necessary EID-to-RLOC mapping is installed on the data plane.

All subsequent packets from Host 1 to this specific EID are sent to the xTR (**h**). Using the EID-to-RLOC mapping in the Map-Cache, packets now get encapsulated and are forwarded at line rate towards the destination (**i**). The destination host stores a timestamp on arrival of a packet.

The experiment is repeated 200 times, each time calculating the latency introduced by each of the stated components by the use of the stored timestamps. The mean values and standard deviations of the results are compiled in Table II. The overall time from the first packet that was sent by Host 1 to the arrival of the first packet at Host 2 is 19.74 ms on average. The mean measured time for steps a-c is 4.45 ms, representing the controller delay for issuing a discovery message to the control plane. The mean measured delay introduced by the control plane (steps d+e) is 12.61 ms. The mean measured controller delay for processing the mapping information from the control plane and installing the EID-to-RLOC mapping in the Map-Cache until the first packet from the source arrives at the destination is 2.58 ms.

E. Controller Performance

We evaluate the performance of the P4-LISP control plane. The setup is the same as the one in Figure 6. We use nping [49] to generate packets destined to different EIDs so that each of them causes a miss in the Map-Cache and requires communication with the controller. The maximum rate of nping is 50,000 packets per second (pps) in our setup. We measured the packet latency for different rates and observed that it increases over time when the packet rate exceeds 150 pps. We therefore conclude that the controller can handle a continuous packet processing rate of 150 pps.

The P4 controller is only a PoC implementation and its performance may be improved. First, the usage of Python as a programming language is not ideal. Languages like C or Rust may increase the packet processing time/rate of the controller significantly. Second, the P4Runtime interface is still under development and subject to changes, which may increase the efficiency of that interface.

VI. Conclusion

In this paper, we presented P4-LISP, an architecture and prototype for a P4-based open-source LISP router that runs on high-performance hardware targets such as the Intel Tofino ASIC. We validated the functionality of the P4-LISP implementation and evaluated the performance of its data plane and control plane. The different router types of the P4-LISP implementation are able to process and forward packets at up to 100 Gbit/s line rate. This holds even in the presence of decapsulation and encapsulation, or double encapsulation. The implementation is expected to run also on newer ASIC generations, such as Intel Tofino II, with a line rate of 400 Gbit/s. A latency analysis revealed that processing times of Map-Requests through the P4 and LISP control plane are about 20 ms; this applies only for rare misses in the Map-Cache. The control plane of the current implementation is able to process 150 Map-Requests per second. As the control plane may be subject to DoS attacks, P4-LISP provides countermeasures like black- and white-listing of source EIDs and RLOCs, and it is able to protect the control plane against overload by limiting the rate of Map-Requests from the data plane. The code for the P4-LISP prototype is available on GitHub [46].

References

[1] B. Feng *et al.*, "Locator/Identifier Split Networking: A Promising Future Internet Architecture," *IEEE COMST*, vol. 19, no. 4, 2017.

[2] A. Cabellos *et al.*, "An Architectural Introduction to the Locator/ID Separation Protocol (LISP)," RFC 9299, 2022.

[3] D. Farinacci *et al.*, "The Locator/ID Separation Protocol (LISP)," RFC 9300, 2022.

[4] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM CCR*, vol. 44, no. 3, 2014.

[5] Intel Corporation, "Intel Tofino 2: P4-programmable Ethernet switch ASIC," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html, 2021.

[6] D. Farinacci, "lispers.net," https://www.lispers.net/, 2021.

[7] F. Baker *et al.*, "Ingress Filtering for Multihomed Networks," RFC 3704, 2004.

[8] D. Lewis *et al.*, "Interworking between Locator/ID Separation Protocol (LISP) and Non-LISP Sites," RFC 6832, 2013.

[9] D. Farinacci *et al.*, "LISP Mobile Node," https://datatracker.ietf.org/doc/draft-ietf-lisp-mn/12/, Internet-Draft, 2022.

[10] ——, "LISP Canonical Address Format (LCAF)," RFC 8060, 2017.

[11] D. Klein *et al.*, "NAT Traversal for LISP Mobile Node," in *ACM Re-Architecting the Internet (ReArch)*, 2010.

[12] V. Ermagan *et al.*, "NAT Traversal for LISP," https://datatracker.ietf.org/doc/draft-ermagan-lisp-nat-traversal/19/, Internet-Draft, 2021.

[13] D. Farinacci, "Simple LISP NAT-Traversal Implementation," https://datatracker.ietf.org/doc/draft-farinacci-lisp-simple-nat/05/, Internet-Draft, 2022.

[14] ——, "Git Repository of lispers.net," https://github.com/farinacci/lispers.net, 2021.

[15] D. Farinacci *et al.*, "Locator/ID Separation Protocol (LISP) Control Plane," RFC 9301, 2022.

[16] P4 Language Consortium *et al.*, "Behavioural Model Version 2 (BMv2)," https://github.com/p4lang/behavioral-model, 2022.

[17] F. Hauser *et al.*, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," *Journal of Network and Computer Applications*, vol. 212, 2023.

[18] R. Atkinson *et al.*, "Identifier-Locator Network Protocol (ILNP) Architectural Description," RFC 6740, 2012.

[19] R. Moskowitz *et al.*, "Host Identity Protocol Version 2 (HIPv2)," RFC 7401, 2015.

[20] E. Nordmark *et al.*, "Shim6: Level 3 Multihoming Shim Protocol for IPv6," RFC 5533, 2009.

[21] K. Sun *et al.*, "LISP-based Hierarchical Service Mobility Management for the Tactical Edge Computing," in *ICTC*, 2020.

[22] T. Balan *et al.*, "LISP Optimisation of Mobile Data Streaming in Connected Societies," *Mobile Information Systems*, 2016.

[23] M. Coudron *et al.*, "Cross-layer cooperation to boost multipath TCP performance in cloud networks," in *IEEE CloudNet*, 2013.

[24] V. Moreno *et al.*, "The LISP Network: Evolution to the Next-Generation of Data Networks," Cisco Press, 2019.

[25] C. Caiazza *et al.*, "Simulating LISP-Based Multilink Communications in Aeronautical Networks," in *OMNeT++ Community Summit*, 2018.

[26] B. Haindl *et al.*, "Ground based LISP for multilink operation in ATN/IPS communication infrastructure," in *IEEE DASC*, 2016.

[27] D. K. Luong *et al.*, "Seamless Handover in SDN-Based Future Avionics Networks with Network Coding and LISP Mobility Protocol," in *IEEE DASC*, 2019.

[28] E. Seo *et al.*, "The Scalable LISP-deployed Software-Defined Wireless Network (LISP-SDWN) for a Next Generation Wireless Network," *IEEE Access*, vol. 6, 2018.

[29] D. Herrmann *et al.*, "Inbound Interdomain Traffic Engineering with LISP," in *IEEE LCN*, 2014.

[30] D. Farinacci *et al.*, "LISP Traffic Engineering Use-Cases," https://datatracker.ietf.org/doc/html/draft-ietf-lisp-te-10, Internet-Draft, 2022.

[31] K. Okada *et al.*, "Oblivious DDoS Mitigation with Locator/ID Separation Protocol," in *ACM CFI*, 2014.

[32] D. Farinacci *et al.*, "LISP for Satellite Networks," https://datatracker.ietf.org/doc/draft-farinacci-lisp-satellite-network/01/, 2022.

[33] L. Iannone, "The OpenLISP Project," http:/http://www.openlisp.org, 2011.

[34] S. Steffann, "Git Repository of PyLISP," https://github.com/steffann/pylisp, 2019.

[35] A. Stockmayer *et al.*, "jLISP: An Open, Modular, and Extensible Java-Based LISP Implementation," in *ITC*, vol. 01, 2016.

[36] A. Rodriguez-Natal *et al.*, "Programmable Overlays via OpenOverlay-Router," *IEEE Communications Magazine*, vol. 55, no. 6, 2017.

[37] Open Networking Foundation, "ONOS," https://opennetworking.org/onos/, 2022.

[38] OpenDaylight Project, "OpenDaylight," https://www.opendaylight.org, 2021.

[39] Y. Han *et al.*, "Design and implementation of LISP controller in ONOS," in *IEEE NetSoft*, 2016.

[40] The Linux Foundation, "Open vSwitch," http://www.openvswitch.org/, 2022.

[41] M. Hoefling *et al.*, "A Survey on Mapping Systems for Locator/Identifier Split Internet Routing," *IEEE COMST*, vol. 15, no. 4, 2013.

[42] M. Menth *et al.*, "FIRMS: A Mapping System for Future Internet Routing," *IEEE JSAC*, vol. 28, no. 8, 2010.

[43] Cisco Systems, Inc., "IP Routing: LISP Configuration Guide," https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_lisp/configuration/15-mt/irl-15-mt-book.html, 2017.

[44] AVM GMbH, "Info for FRITZ!Box 3272," http://download.avm.de/fritzbox/fritzbox-3272/deutschland/fritz.os/info_en.txt, 2016.

[45] LANCOM Systems GmbH, "LANCOM Routing and WAN Connections: LISP," https://www.lancom-systems.de/docs/LCOS/Refmanual/DE/topics/LISP.html, 2019.

[46] "P4-LISP Prototype Repository on GitHub," https://github.com/uni-tue-kn/P4-LISP.

[47] Edgecore Networks, "Wedge 100BF-32X," https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335, 2021.

[48] The P4.org API Working Group, "P4Runtime: Specification and Implementation," https://github.com/p4lang/p4runtime, 2021.

[49] Nmap.org, "Nping - Network Packet Generation Tool," https://nmap.org/nping/.