Decoupling State from Control in Software-Defined Networking

Alberto Rodriguez-Natal

Advisor: Albert Cabellos-Aparicio, PhD

Co-Advisor: Fabio Maino, PhD

Department of Computer Architecture
Technical University of Catalonia

This dissertation is submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

July 2016

To my parents, to my sister
and to Andrea,
of course

“This is how you do it: you sit down at the keyboard
and you put one word after another until it’s done.
It’s that easy, and that hard.”
- Neil Gaiman

Acknowledgements

I have always thought that the acknowledgments section is the most important part of a thesis for two reasons. First because you can find the research elsewhere, but you can only find the acknowledgments here. Second because while the research evolves and the results get eventually obsoleted, the people you meet along the way remain. Therefore, I have been thinking on what to put in this section for a long time. However, now that I am finally writing it down I believe that I am making no justice to the people mentioned here. I can not find the words to express the gratitude I feel towards them. What follows is only a humble attempt.

As it should be, the first lines are for my advisor, Albert Cabellos. Exceptional researcher, mentor and individual. With so many PhD advisors out there, I consider myself fortunate for having ended up under his supervision. His clear vision, endless motivation and pragmatism drove me through the most challenging periods of this thesis. Thanks also for always giving me honest feedback, even when I was not going to like it. However, for what I must thank him most is for his advice beyond what concerns research. His down-to-earth recommendations and life pro tips came really handy countless times. For those I am as thankful as for his research guidance. Thanks also for being a proficient geek with whom exchange obscure references on movies and comics. Unfortunately, at the time of this writing, determining who outperforms the other on geek knowledge requires further investigation and remains as an open question.

Closely after comes my deep gratitude to my co-advisor, Fabio Maino. My honest appreciation to him, who believed in my ability from the very beginning and encouraged me with sound trust even when I hesitated. Thanks for being a guide to the inner dynamics of the industry and for sharing curated knowledge with a newcomer. Thanks also for periodically giving me the chance to showcase my work to all sorts of audiences, but specially for doing so with solid confidence in the successful outcome. I will be eternally indebted to Fabio for sponsoring my research and for offering me the opportunity to grow professionally, both during my thesis and afterwards. Thanks also for that conversation we once had in our way to the airport. I remember vividly the counseling I received that day.

Let me also express my appreciation to the faculty members of the CBA research group at UPC. First, thanks to professors Jordi Domingo-Pascual and Josep Solé-Pareta, for guiding me during my first steps as a researcher and for having been a source of advice since then. Thanks also to Davide Careglio, for the shared sense of humor and for taking me in those trips to my first project meetings. Finally, thanks to Pere Barlet for his always clever analysis and for the quality feedback provided at different points of my research.

My deep gratitude goes too to some members of the industry that inspired part of this thesis. First of all to Sharon Barkai, for being a modern *patron of the arts* and the greatest expert I know on combining networks and databases. My great appreciation to Sharon for completely changing my view on SDN and for being a great source of inspiration for this thesis. Sharon was the one who coined the idea of "global knowledge, local decisions", and Chapter 4 was only possible thanks to him (kudos also to the rest of the great guys from ConteXtream: Ariel Noy, Ajay Sahai, Gideon Kaempfer, and the others). My most special thanks also to Dino Farinacci, one of the *fathers* of LISP and engineer beyond compare. Capable, like no one else, of discussing both high-level architectural abstractions and bit-level details of the implementation. I greatly appreciate those technical discussions on all things LISP, and in general all the help provided over these years. Finally, my honest gratitude to David Meyer, truly visionary in the networking field, for sharing his vision. Thanks to Dave for always being glad to engage on discussions on the future of networking, and for giving me a hand whenever I needed it.

My sincere appreciation also to those LISP experts at Cisco. First and foremost to Vina Ermagan, one of the major driving forces of this thesis. Vina's talent for team building and gracious management were hugely appreciated when we were struggling to make it through deadlines. I could not thank her enough for these years of tireless support and guidance, on this thesis and beyond. Her hard work and sharp advice truly shaped this research. My great gratitude as well to Marc Portolés, who saved me on my first visit to the bay area and made the stay much more enjoyable. For that, I owe him countless beers. Thanks also for being such a passionate researcher and for dragging me into those creative technical discussions. Finally, a huge thank you to Darrel Lewis for his always realistic technical advice, to Preethi Natarajan for those early days of research on LISP-MN and to Vasileios Lakafosis for all his help with LISPmob.

I could not go without mentioning those months spent in Tokyo working at the laboratory of Professor Yusheng Ji. Among all the great experiences that this thesis has granted me, I remember my stay in Japan as one of the best. My deep gratitude to Prof. Ji as well for giving me a different perspective on my research. Her deep knowledge and multidisciplinary experience truly completed my work. Thanks also to Kien Nguyen for his practical advice on how to survive in Tokyo and for those discussions about OpenFlow in the coffee room. Thanks to both for keep supporting me in the present day, long after my internship ended.

A special note also to those I met via the IETF. A huge thanks to Luigi Iannone and Damien Saucez for the discussions on LISP and for making those IETF meetings more interesting. Thanks as well to Joel Halpern for sharing his technical wisdom and enlightening me on the intricate mechanisms of the IETF. Thanks also to Diego López for sharing his perspective on SDN, and for -unbelievably- always being present in all conferences I go.

A final mention to the people who *lived* with me in the D6-008 office. First to Florin Coras, who is ultimately the one to blame for this thesis. Shall I had not listened to him, I would have not ended up doing a PhD with Albert and Fabio. Fortunately enough, I trusted his word back then. Sorry for all the silly jokes during these years Florin, it is just my way to pay you back. Thanks as well to Loránd Jakab, who guided me during the beginning of my research. He gave me right on the spot technical advices at that time and continues to do so today. Thanks also to Albert Mestres, Sergi Abadal, Raül Gómez and Valentín Carela for the lunches and the fun. I could not ask for better people to share an office with. Finally, my most deep and sincere gratitude to Albert López, an extraordinary research engineer but also a good friend. His rigorous work, methodical testing, and continuous contributions (often behind the scenes) can not be thanked enough. May these lines serve to give him part of the credit he deserves. *Gràcies Albert!*.

To conclude, I would like to acknowledge the funding that made this thesis possible. Thanks to Cisco Systems for their generosity supporting this research and to the Spanish Ministry of Education, Culture and Sport for supporting me trough scholarship FPU2012/01137.

A nivel personal me gustaría agradecer a mis compañeras de piso, Raquel y María, por aguantarme estos últimos años y en especial estos últimos meses. Sé que a veces no ha sido fácil. Gracias también a toda esa gente que lleva años *on fire* y a todos los presentes (físicamente o no) en ese 28 de mayo. A todos ellos perdón por no haber podido estar en cafés, partidas y cervezas. Sabéis que os quiero a todos, pero la tesis siempre fue muy celosa.

Quiero dar las gracias de manera especial a mi familia. A mis padres, por darme todo su apoyo durante estos años, pero sobretodo porque nunca dudaron. Por ellos he llegado hasta aquí y por ellos ahora tengo que seguir adelante. Perdón por irme, pero gracias por entenderlo. Gracias también a mi hermana Laura, por todas las llamadas que le debo. Siento que hayas tenido que ser *hija única*, pero estoy orgulloso de como has sabido salir adelante.

Las últimas palabras, por supuesto, son para Andrea. Para ella, a quien no hizo falta explicarle lo que era el índice JCR. Para ella, por todas las noches que tuvo que dormir con la luz encendida. Para ella, por estar tan loca como para acompañarme en la aventura que empieza donde acaba este doctorado. Para ella, gracias.

Abstract

Software-Defined Networking (SDN) arose as a solution to address the limitations of traditional networking. In SDN networks, the control-plane is decoupled from the data-plane devices and logically centralized in a new network element, the SDN controller. SDN enables easier network operation and allows forwarding devices and control logic to evolve independently. The centralization of the control permits to have a global view of the network and act on it as a whole, but at the same time requires a careful design to keep the controller scalable. Commonly, a logically centralized controller is instantiated over a physically distributed infrastructure that leverages on a distributed network state database. Control applications running on top of the controller modify this state to make it compliant with their control policies or to react to network events. The controller programs the data-plane devices to reflect these state changes.

Interestingly, current SDN approaches keep the network state architecturally as part of the controller. However, this thesis arguments that the network state can be an SDN component on its own, logically separated from the controller. In the same way that originally SDN decoupled control from data, this thesis lays the foundations to explore the decoupling of state from control. This logical separation entitles state and control to scale independently and allows focusing on their individual functionality and requirements. This may be beneficial, at least, when the control has to be asynchronous and when the control has to be decentralized. For those scenarios this thesis describes two architectures driven by specific use-cases.

On one hand, when data-plane devices are subject to a high churn they require an asynchronous control communication with the controller. This is the case for end-nodes (e.g. smartphones, home-routers) since they are transient and/or highly mobile. In this case, pushing the state to the data-plane devices presents an architectural challenge. As a consequence, to enable SDN for end-nodes we advocate for a design where the state is rather pushed to a standalone database disjointed from the controller. Data-plane devices directly access this state database and retrieve the state they need on demand. Following this idea, we propose an SDN architecture that leverages on distributed and symmetric controller nodes offering an intent-driven northbound to the control applications, and on a state database with a connectionless pull-based southbound towards the data-plane nodes.

On the other hand, SDN centralization comprises several challenges besides keeping the controller scalable. The control signaling required introduces an inherent latency burden and the aggregation of local information conceals local details. Therefore, SDN centralization may result unsuitable for scenarios that require fine local control with minimal latency. This is the case of Network Function Virtualization (NFV) in operator networks. For that scenario this thesis describes an architecture where the state remains centralized, but the control is decentralized and moved close to the data-plane devices. The architecture seeks to find a balance among the traditional decentralized networks and the centralization brought by SDN. In contrast to existing SDN deployments, the control is distributed over the network but federated and coordinated thanks to the central state database.

In both described architectures we use the Locator/Identity Separation Protocol (LISP) for state exchange. Therefore, another contribution of this thesis is to analyze LISP as an SDN protocol. Besides, in the second part of the thesis we delve deeper into the implications of deploying SDN for end-nodes. Particularly, we analyze the mobility aspects of LISP signaling along with its inherent privacy concerns and we introduce OpenOverlayRouter, a LISP-capable overlay software for end-nodes SDN deployments.

Resumen

Las redes definidas por software (SDN) aparecen como solución a las limitaciones de las redes tradicionales. En SDN el control se extrae de los dispositivos del plano de datos y se centraliza a un nuevo dispositivo llamado controlador. La centralización del control permite tener una visión y gestión global de la red, sin embargo el controlador se ha de diseñar con cuidado para que sea escalable. Normalmente, un controlador centralizado lógicamente se despliega sobre una infraestructura distribuida físicamente, en parte haciendo uso de una base de datos que almacena el estado de la red. Las aplicaciones de control que se ejecutan sobre el controlador modifican este estado conforme a sus políticas de control o como reacción a eventos en la red. En respuesta, el controlador programa el plano de datos para reflejar estos cambios en el estado.

Las propuestas SDN existentes consideran arquitecturalmente el estado como parte del controlador. Esta tesis, sin embargo, defiende que el estado de la red puede ser un elemento por si mismo, separado del controlador. De la misma manera que originalmente SDN separó el plano de control del plano de datos, esta tesis abre el camino para explorar la separación de estado y control. Esta separación conceptual hace posible escalar estado y control por separado y permite centrarse de manera individual en las funcionalidades y requerimientos de cada uno. Esto sirve de ayuda cuando el control tiene que ser asíncrono y/o cuando el control tiene que ser descentralizado. Para esos dos escenarios, esta tesis describe dos arquitecturas motivadas por casos de uso concretos.

Por un lado, cuando los dispositivos del plano de datos no están siempre disponibles, necesitan comunicarse con el controlador de manera asíncrona. Este escenario se da con dispositivos de red finales (móviles, routers domésticos, etc) que se conectan transitoriamente a la red y/o cambian de conexión con frecuencia. Este escenario dificulta que el controlador programe de manera pro-activa el estado en estos dispositivos. Así pues, para integrar estos dispositivos en despliegues SDN, esta tesis aboga porque el controlador almacene el estado en una base de datos independiente, separada del controlador, a la que los dispositivos acceden directamente para obtener el estado que necesiten cuando lo necesiten. Siguiendo esta idea, proponemos una arquitectura SDN para dispositivos finales basada en un controlador distribuido con una interfaz declarativa hacia las aplicaciones de control y en una base de datos con una interfaz sin conexión y bajo demanda hacia el plano de datos.

Por otro lado, la centralización de SDN presenta varios desafíos más allá de la escalabilidad del controlador. En concreto, la señalización de control requerida introduce una latencia adicional y la agregación de la información oculta los detalles locales. Esta centralización resulta inadecuada cuando se necesita un control local preciso con mínima latencia. Este es el caso de la virtualización de funciones de red (NFV) en redes de operadores. Para ese escenario esta tesis describe una arquitectura donde el estado permanece centralizado pero el control se descentraliza y mueve cerca del plano de datos. Se busca equilibrar la descentralización de las redes tradicionales y la centralización de SDN. En contraste con los despliegues SDN existentes, el control está distribuido por la red pero federado y coordinado gracias a la base de datos central.

En las dos arquitecturas descritas usamos el Protocolo de Separación de Localización e Identidad (LISP) para el intercambio de estado, por tanto otra contribución de esta tesis es analizar LISP como protocolo SDN. En la segunda parte de esta tesis profundizamos en las implicaciones de desplegar SDN para nodos finales. Particularmente, analizamos LISP en entornos de movilidad junto con su problemática en términos de privacidad y presentamos OpenOverlayRouter, un software para despliegues SDN basados en LISP.

Contents

List of Figures

List of Tables

Nomenclature

Roman Symbols

API	Application Program Interface
AS	Autonomous System
BGP	Border Gateway Protocol
DFZ	Default Free Zone
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DPDK	Data Plane Development Kit
ETSI	European Telecommunications Standards Institute
ForCES	Forwarding and Control Element Separation
GPE	Generic Protocol Encapsulation
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
JNI	Java Native Interface
LAN	Local Area Network
LISP	Locator/ID Separation Protocol

LTE	Long-Term Evolution
MAC	Media Access Control
MANO	Management and Orchestration
MN	Mobile Node
MPLS	Multiprotocol Label Switching
MR	Map Resolver
MS	Map Server
NAT	Network Address Translation
NFV	Network Function Virtualization
NIB	Network Information Base
NVGRE	Network Virtualization Using Generic Routing Encapsulation
OOR	OpenOverlayRouter
OSPF	Open Shortest Path First
OVS	Open vSwitch
PCE	Path Computation Element
PxTR	Proxy Ingress/Egress Tunnel Router
RTR	Re-encapsulating Tunnel Router
SDN	Software-Defined Networking
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtual Network Function

VxLAN	Virtual Extensible Local Area Network
WAN	Wide Area Network
WLAN	Wireless Local Area Network
xTR	Ingress/Egress Tunnel Router

Chapter 1

Introduction

1.1 Background: Legacy networks

Computer networks have become today an element of utter importance. Nowadays, computer networks are present in almost all the systems that we interact with on daily basis. The rise of the Internet over the last decades as a world-wide network publicly accessible to everyone has shifted the way that people work and interact. However, reaching this point has been an effort of years of research from individuals, companies and institutions.

The development of packet switching networks over the decade of the 60s was the germ to the computer networks the way we know them today. The need of researchers to interconnect computers to enable faster information exchange, long distance communications and no single point of failure made that different efforts evolved in parallel during the 60s and 70s. Those efforts, led by different individuals across the world, ended up in designing different methods and protocols to interconnect computers. Eventually, the different communication proposals converged into the family of protocols most widely used today, the TCP/IP stack (Transport Control Protocol / Internet Protocol) [115, 116]. With the advent of TCP/IP as the *de facto* standard for computer interconnection it was possible for computers from different manufactures and software from different developers to exchange information packets using common protocols that both sides understand.

However, computer networking not only requires to enable computers to talk with each other, but also to deploy and manage a networking infrastructure to carry those packets that the computers exchange. Computer networks are then composed of packet forwarding elements that move the packets from one end point to another. Those forwarding elements comprise several ports and interfaces that link them to other forwarding elements or to end point computers. Packets are clearly labeled with the addresses of the sender and receiver in order to make possible to deliver them to the correct destination computer. In order to take a path through the network that eventually leads to the packet destination, the forwarding elements must know which of their available ports they should forward the packet through. Commonly in computer networks, the part of a networking device that receives and forwards packets is known as the data-plane while the part that is in charge of learning and deciding how to forward those packets receives the name of control-plane.

In order to find suitable paths to forward the packets, the control-plane at the networking elements relies on different control protocols. In the effort to offer no single point of failure, these protocols operate decentralized across the different forwarding devices. Control protocols usually implement distributed algorithms derived from graph theory. For instance, in Local Area Networks (LAN) leveraging on Ethernet technology [92] it is common to implement the Spanning Tree Protocol (STP) [110] that finds a loop-less tree within the network topology. In large IP-based networks under a single administrative domain the most common approach is to use a link state routing [88] protocol. Link state protocols leverage on the idea that each forwarding element discovers and announces its neighbors to the rest of forwarding elements. Using variations of the Dijkstra algorithm [17] on that data, each forwarding element is able to build a map of the network and compute the shortest path per each destination. Notable examples of link state protocols are Open Shortest Path First (OSPF) [95] and Intermediate System to Intermediate System (IS-IS) [9]. Finally, to interconnect IP networks under different administrative domains the most widely used protocol is the Border Gateway Protocol (BGP) [120]. In BGP each border router connecting to other administrative domains announces the networks reachable through its domain and in exchange it learns the networks reachable through the domains of its neighbors.

The described control scheme made the nature of computer networks distributed and decentralized. At the same time, protocol standardization ensured the correct inter-operation of different implementations and devices. These two factors combined made possible an organic growth of the network without the need for a central authority. For the first time in history, it was feasible to deploy and manage large computer networks and to interconnect networks of different administrative domains. Thanks to this decentralization and protocol standardization, computer networks were subject to a rapid spread and expansion. However, in an effort to remain interoperable and backwards compatible with existing deployments, the core network protocols (TCP, IP, OSPF, BGP, etc) have seen no fundamental change for the last decades. Although evolutionary enhancements and new protocols have appeared over the years, the foundations of networking have remained unmodified to the present day. This has come to be known as network *ossification*.

Furthermore, the factors that allowed rapid growth and expansion in the early days of networking came with a set of inherent drawbacks. The distributed nature of the protocols require to individually configure each network device. In many cases this has to be done manually by the network operator. Moreover, this decentralization prevents from having a global view of the network, which greatly increases the complexity of debugging and optimization. Although some solutions today enable a partially centralized management, the network itself still operates as a combination of different independent distributed protocols.

Besides, any new protocol to be deployed requires of years of refinement and interoperability testing to ensure compatibility with deployed networks. Even when a new protocol has been well-tested and proven to be useful, deploying it in the network is not a trivial task. On one hand, data-plane protocols (such IP) are in most cases implemented in the hardware of forwarding devices, and thus any new protocol or protocol update involves to individually upgrade the boxes across the network. On the other hand, control protocols, that usually run in software and therefore should be easier to upgrade and replace, are still mostly unmodified. The fact that control protocols run in a distributed manner within the networking boxes makes any modification to them a challenge since many boxes need to be upgraded before the protocol can be put into use.

Due to network ossification, all the described challenges are still present in current networks today. For a long time, researchers have been looking into ways of dealing with the ossification of the network and of facing the inherent problems of the current networking paradigm. Nevertheless, in the later years it has arose a solution that has gained major traction and interest from both academia and the industry, Software-Defined Networking (SDN).

1.2 State-of-the-art: Software-Defined Networking (SDN)

Software-Defined Networking (SDN) arose as a solution to deal with the limitations of traditional networking. Although there are different definitions for SDN depending on the focus and scope [48], [35], [61], it is generally agreed that a SDN solution comprises two main characteristics. The decoupling of data and control planes and the centralization of network control. This is possible through the use of well-defined interfaces between the control-plane and data-plane elements, which in turn enables remote network programability.

In SDN networks, the control-plane is extracted from the data-plane devices and pushed to a new network element, the SDN controller. This controller centralizes the network control in a single logical point and via a southbound interface it remotely programs the forwarding rules at the data-plane devices. The decoupling that SDN introduces allows data and control planes to evolve independently. Data-plane research can focus on improving the forwarding performance and on ways to better expose the forwarding internals to the controller while the control logic is implemented as software running on the controller. The control software instructs the controller on how it should program the data-plane and decides what to do when the data-plane receives new or unexpected traffic.

1.2.1 OpenFlow Protocol

The research community generally agrees on idea that the current SDN trend started with the OpenFlow [87] protocol developed at the Stanford university. According to Feamster et al. [35] the term *Software-Defined Networking* was first coined in a MIT (Massachusetts Institute of Technology) press article [45] about OpenFlow where the author stated:

> Frustrated by this inability to fiddle with Internet routing in the real world, Stanford computer scientist Nick McKeown and colleagues developed a standard called OpenFlow that essentially opens up the Internet to researchers, allowing them to define data flows using software - a sort of "software-defined networking".

However, the ideas of decoupling control from data or centralizing the network control were not first mentioned with the OpenFlow protocol. On one hand, the trend on *active networking* [135] was already discussing network programmability in the decade of 1990. On the other hand, proposals like Path Computation Element (PCE) [34] or Forwarding and Control Element Separation (ForCES) [20] respectively offered centralized control or decoupled data and control planes in the 2000s. Moreover, the work on the Click modular router [68] in 2000 explored the potential of flexible, modular and highly configurable data-plane devices. Current SDN is the result of a long history of research from different authors. The OpenFlow specification itself builds on top of ideas from just prior works like 4D [44] and Ethane [12]. For a summary of the research history that led to SDN we recommend the work of Feamster et al. in [35].

Despite some misconceptions on the topic, SDN as commonly understand today is not constrained to the OpenFlow protocol. Although OpenFlow is probably the most notorious example of a SDN southbound protocol, it is not the only one. The definition and scope of SDN varies across the research community and in that sense, industry and academia are experimenting with different SDN protocols. Depending on the use-case and scenario other protocols can be deployed, sometimes in parallel with OpenFlow. At the time of this writing, and without trying to be exhaustive, other notable examples of protocols used for SDN are: the Forwarding and Control Element Separation (ForCES) Protocol [20], the Path Computation Element (PCE) Communication Protocol (PCEP) [139], Locator/ID Separation Protocol (LISP) [30], the Border Gateway Protocol (BGP) [120] and the Network Configuration Protocol (NETCONF) [24]. It is worth to note that some of the protocols used today for SDN were developed prior to OpenFlow appearance and even before the term SDN was coined.

For reference and due to its importance in the evolution of SDN, in what follows we discuss briefly the OpenFlow architecture. In its very basic form [87, 52], an OpenFlow switch has a flow table and an OpenFlow channel towards the SDN controller. Flow tables contain flow rules, each of them is composed by a set of matching fields (e.g. destination IP address, VLAN tag, etc) and the actions to perform in the packet (e.g. modify source Ethernet address, forward packet through a specific port, etc). When an incoming packet is received, it is matched against the flow table and the appropriate actions are taken. If no suitable rule is found (i.e. the packet matches no rule), the switch has to ask the controller what to do with the packet. It does so by means of the *packet_in* OpenFlow message which sends the packet to the controller. The controller can then insert a new rule in the flow table (or modify an existing one) using the *flow_mod* message. In many cases, the controller has proactively programmed the OpenFlow switch using *flow_mod* messages rather than waiting for *packet_in* messages to happen.

1.2.2 Controller Evolution

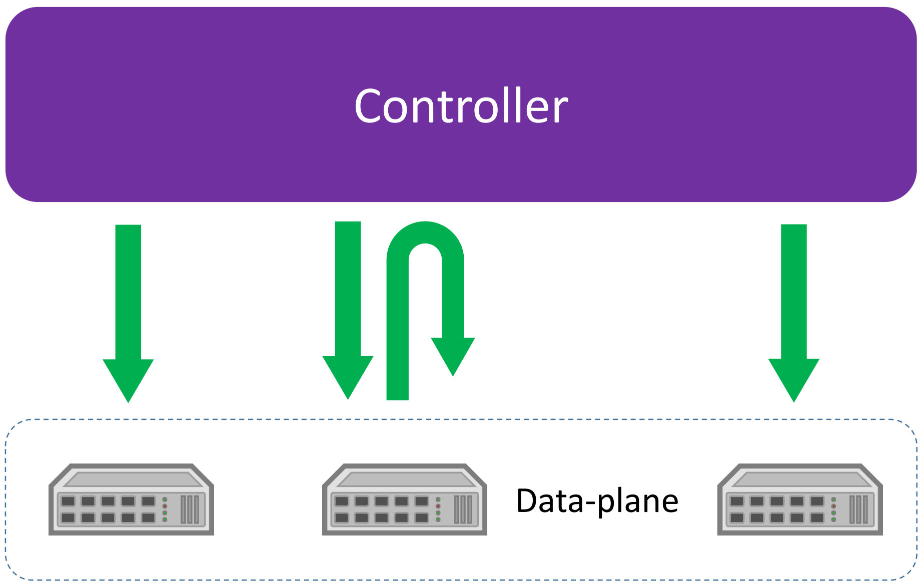


Figure 1.1 **Original SDN architecture**

The early works on OpenFlow focused on the interface to program the data-plane devices, that is, in the specification of the OpenFlow protocol. At that time the interest was in the signaling messages required between the controller and data-plane elements and in how the former should expose their internal forwarding tables. The first conceptual ideas of SDN, as imagined by the original OpenFlow article [87], devised a SDN architecture as the one depicted in Fig. 1.1. In those early architectures there was non distinction between the controller and the actual software pieces that implemented the control logic. Control applications where part of the controller itself.

Building on top of the OpenFlow architecture, the research community started to explore and refine the SDN controller. One of the first controller designs to appear was NOX [47], which presented the controller as an operating system for networks. NOX introduced the idea of the controller as a mere *programmatic interface* to the network where the applications implemented on top were the ones in charge of the actual network control. Today, the interface that the controller exposes to the software control applications is usually referred as the northbound interface. The authors of NOX proposed one of the first northbound SDN interfaces, the Flow-based Management Language [54]. That interface was still rather simple (and strongly influenced by the OpenFlow protocol) but provided enough abstraction to enable developing of applications on top. NOX also introduced another key concept on modern SDN, a database storing the global view of the network. NOX used this database to store observations made from the network, such the topology of switches, the location of hosts and middleboxes, the services offered in the system, etc. Via the controller, the northbound control applications used the state stored in the database to make control decisions.

As the interest and research on SDN continued over the years, it was clear that this new paradigm opened new possibilities for networking but also brought new challenges. The centralization of the control allows to keep a global view of the network and act on the network as a whole, but at the same time requires of carefully design to keep the controller scalable. In that sense, both the academia and the industry have discussed different implementation approaches to scale the SDN controller. Most of them comprise a distributed controller that is logically centralized but supported by different physical nodes. Such controllers use a distributed database to store the global network state. One of the first and most well-known proposals of a distributed controller is Onix [71].

Onix addresses the scalability requirements of the controller by enforcing a strongly distributed approach. Onix is composed of different controller instances running across different servers. These servers also store partitions of a distributed database where Onix stores the network state. A major contribution of Onix to the SDN field is the introduction of the term Network Information Base (NIB). The NIB refers to the network state stored at the database that provides detailed information on the network. Compared to NOX, Onix also improves the northbound interface offering a more general API to northbound applications. Onix has inspired important following research on distributed controllers such B4 [59] and ONOS [5]. B4

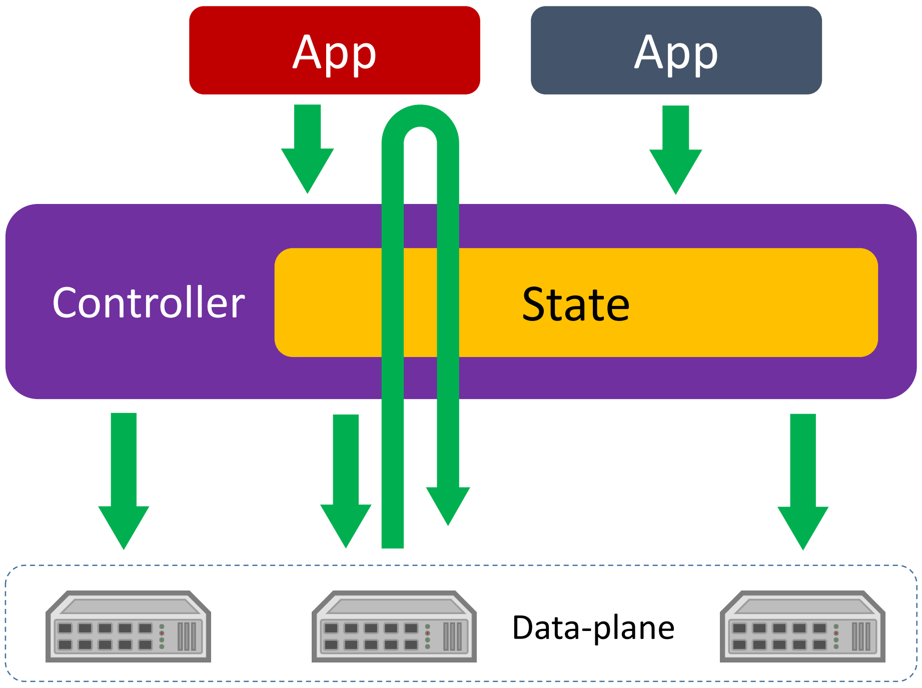


Figure 1.2 **State-of-the-art SDN architecture**

proposes a decentralized controller to control a large worldwide WAN network. Building on the steps of Onix and B4, ONOS (Open Network Operating System) represents the state-of-the-art on distributed SDN controllers. ONOS offers an optimized architecture that leverages on the latest advancements on distributed systems and distributed databases to offer high performance, low latency and great scale-out capabilities.

Besides ONOS, the OpenDaylight [89] controller represents an excellent example of a modern controller, cluster-oriented and with support for several different northbound and southbound protocols. The general architecture of modern controllers (such ONOS or OpenDaylight) is depicted in Fig. 1.2. As the figure suggests, network applications running on top generate control policies through the northbound interface. These policies are usually expressed by means of an intent-driven language (e.g. declarative) that has to be latter processed by the controller. Examples of such declarative northbound interfaces can be found in [36, 66, 93]. The controller renders these abstract directives into specific data-plane rules making use of the network state that is stored in the state database. The data-plane specific state is then pushed to the data-plane devices via a southbound protocol. Despite the reactive (e.g. pull-based) early works on SDN [12, 47], most of the controller designs today heavily rely on proactively pushing the state to the data-plane elements in advance.

In general, the current state-of-the-art shows how the controllers are designed to cope with the scalability requirements of modern SDN deployments. However, achieving the required scalability imposes several distribution tradeoffs that controllers need to balance and that are well-documented in the research literature [76, 53, 112, 10, 131, 130, 19, 5, 71]. This makes the design of SDN controllers a non-trivial and complex endeavor. Moreover, the design complexity is expected to rise as SDN moves to new scenarios and has to face additional challenges. In this thesis we wonder if it is possible to follow a different approach when designing SDN controllers.

1.3 Motivation: Decoupling State from Control

Interestingly, all current SDN approaches keep the network state architecturally as part of the controller. The network state, network information base or global network view (depending on the controller terminology) is always kept within the logical boundaries of the controller. In terms of deployment this does not constrain the back-end database storing the state to be physically allocated within the same machines that host the controller instances, although this is a common scenario. In existing SDN proposals, the state is not directly exposed to the northbound applications or the southbound data-plane devices, rather southbound events or northbound updates go through the controller, which is then in charge of modifying the state.

However, there are scenarios with characteristics that would advise to use a design that takes the state logically out of the controller. This thesis arguments that the network state can be a SDN component on its own, in the same terms than the controller, the northbound applications or the data-plane devices. In the same way that SDN originally decoupled control from data, this thesis lays the foundations to explore the decoupling of state from control. It should be noted that this thesis does not propose to create the entity of the network state. Such entity is already well-defined (under different names) on SDN literature. Similarly to the dissociation of controller and control applications, that led to the definition of the northbound interface, this thesis does not introduce the state, but rather defines it as a different entity dissociated from the controller.

This logical separation entitles state and controller to scale independently. This allows to focus on their different individual requirements, since the set of skills required to design a scalable controller are different from those needed to design a scalable state database. Moving the state to a disjoint database moves along many of the scalability requirements of the controller to this database. Therefore, the scalability of the SDN system is no longer tied to the scalability of the controller but rather to the scalability of the state database. To scale the database there is no need to reinvent the wheel and instead all the research and advancements on the field of

-distributed- databases can be directly leveraged. Furthermore, state decoupling removes or minimizes the requirement for an east-west interface across different controller instances. The requirement of inter-controller communication was mostly due to the need to distribute and coordinate the state across instances. However, when the state is moved to an external entity, the distribution task is handed to the back-end database and thus the coordination among controller instances alleviated. In general, the state-decoupling eases the design of SDN architectures and particularly of those that require of special interaction with the network state.

1.4 Objectives: Decoupled-State Architectures

Trying to put the state-control decoupling into context, this work describes particular SDN scenarios where a decoupling of state and control would be more effective than keeping the state within controller boundaries. Furthermore, it describes SDN architectures, designed for those scenarios, that take advantage of state-control decoupling. This work identifies two cases where a decoupled-state approached would be beneficial, when the control has to be asynchronous and when the control has to be decentralized.

1.4.1 Asynchronous Control

There are scenarios where data-plane devices are subject to a high churn. In such scenarios, pushing the data-plane state to the data-plane devices presents an architectural challenge. For those scenarios this work advocates for keeping a standalone data-plane state database, architecturally disjoint from the controller. In such proposal, the state is pushed to the database and then retrieved on demand by the data-plane devices. Contrary to existing SDN solutions, the data-plane devices directly access and retrieve the state rather than being the controller the one that push the state to them. Data-plane nodes also push the data-plane events directly to the state database, instead of notifying the controller.

This leaves the controller almost stateless and mostly as a renderer of northbound polices into data-plane state. The northbound applications will define the data-plane policies in advance and pass them to the controller through the northbound interface. The controller will then render those policies into specific data-plane state, based on the state that it sees on the state database, and then store rendered state in the database. Northbound applications periodically analyze offline the stored network state (since it may contain new events from the data-plane) and update northbound policies as they find appropriate.

An example of an asyncronous decoupled-state architecture is depicted in Fig. 1.3. Since the state can not be pushed to the data-plane, asynchronous state architectures are reactive to the

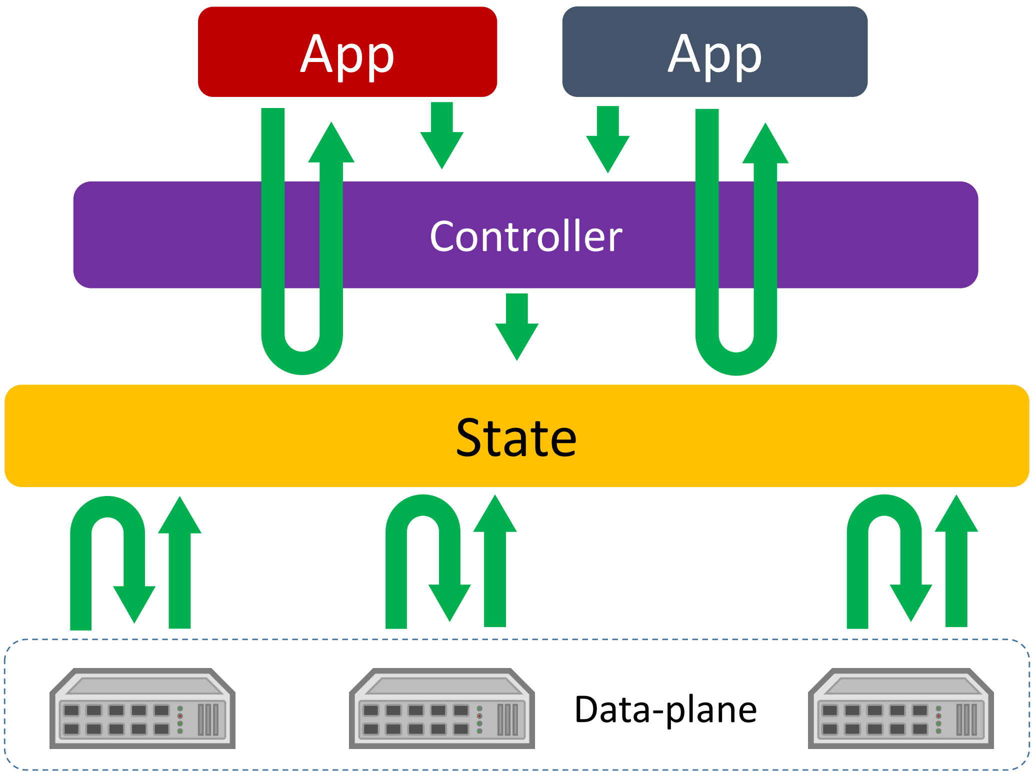


Figure 1.3 **Asynchronous SDN architecture**

data-plane traffic. Similarly to Ethane [12] and NOX architectures [47], they also rely on the first packet of each flow to trigger the retrieval of state from the controller. A decoupled-state architecture for asynchronous communication will greatly rely on that mechanism and thus require state exchange protocols optimized for a pull-based model.

OpenFlow allows to ask the controller for the rules to handle a particular flow or send packets to the controller for inspection via the *packet_in message*. The main motivation for such mechanism was to cover the cases where the data-plane lacks the rules to process certain incoming traffic. In that sense, OpenFlow is a candidate to be used to reactively program the data-plane devices as response to the observed data-plane traffic. Other option that we analyze in this thesis is to use the LISP protocol since it was specifically designed to quickly retrieve and exchange operational state via pull-based mechanisms.

Furthermore, the scenarios where the control communication with the data-plane has to be asynchronous may not benefit from a protocol that requires maintaining a connection established with the controller. Data-plane devices will likely be mobile or transient which will lead to frequent connection reestablishment. In our work we leverage on LISP since it is already connectionless, but another option is to use connection-oriented protocols that only established short lived connections, such gRPC [46].

1.4.2 Decentralized Control

In general, it is agreed that the control centralization brought by SDN enables many benefits, but also comprises several drawbacks. On one hand, keeping the centralized controller scalable is an architectural challenge. Besides, there is an inherent latency burden imposed due to the inter-controller signaling involved. Part of the research community is starting to wonder if the required distribution of the controller would not bring the same problems that the original distribution of network control brought. Although this is arguable, it raises a valid concern. At the time of this writing it is still too early to evaluate the long time consequences of distributed controllers.

On the other hand, the centralized design aggregates and combines local information which may conceal local details. Current centralized approaches may result infeasible in scenarios that require fine grain control of local traffic while remaining performance efficient and scalable. For instance, we believe that Network Function Virtualization (NFV) may be one of those scenarios. Some NFV use-cases may need a fine local control of the traffic while still being globally orchestrated. NFV is a rising trend in the industry and examples of it are the ETSI NFV framework [27] or the ETSI-hosted Open Source MANO (OSM) [101] project.

To cope with the decentralization concerns and to allow fine-grain local control in the scenarios that require it, this thesis describes the idea of a decentralized SDN architecture with centralized state. A decoupled-state architecture that keeps the state centralized but pushes the controllers close to the data-plane devices they control, as depicted in Fig. 1.4. In contrast to existing SDN, the control is distributed over the network, but federated and coordinated thanks to the shared state database. Such architecture seeks to find a balance among the traditional decentralized networks and the new centralized approaches brought by SDN. Despite the decentralization of the control, it is easier to manage decentralized decoupled-state networks when compared to classical decentralized networks since it is still possible to obtain a global view of the network from the centralized state.

In decentralized decoupled-state architectures, the global state is partially cached locally at the controllers. This makes a difference with the case of asynchronous decoupled-state architectures, since the controllers of the decentralized architectures are not stateless. Furthermore, to control their local data-plane devices decentralize controllers also generate local state that is only used locally by the local controller and not pushed to the global state database.

In the context of SDN, it should be noted that a decentralized control is not the same as a distributed control. While most SDN proposals today comprise distributed control to cope

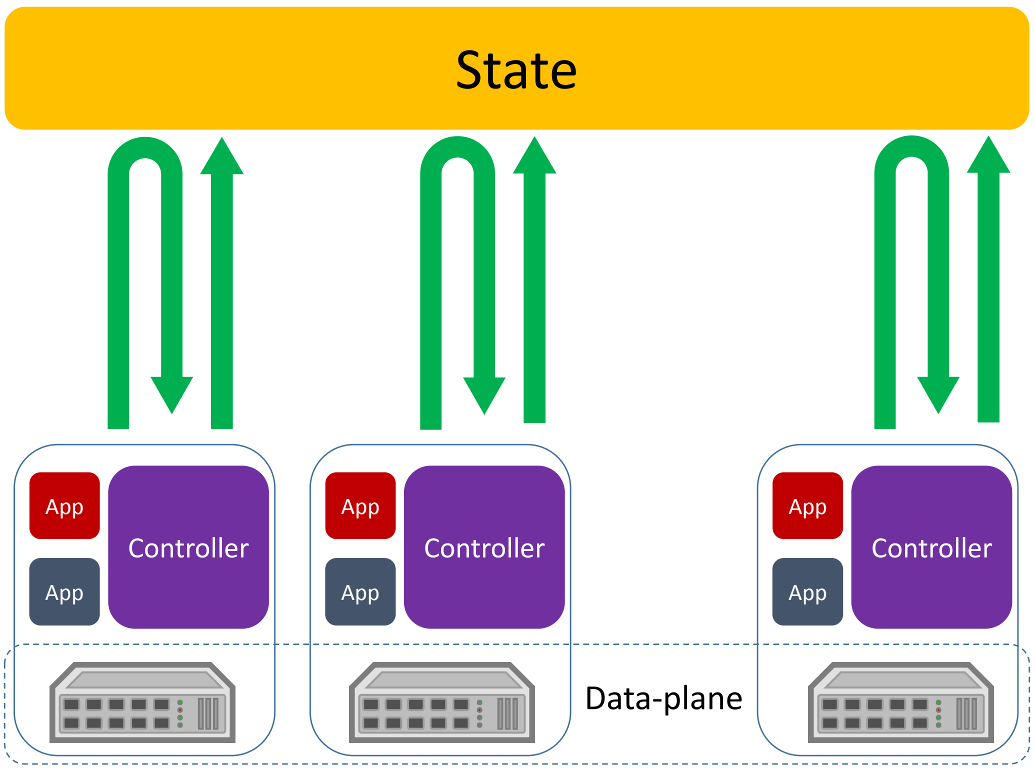


Figure 1.4 **Decentralized SDN architecture**

with scalability requirements [130, 19, 11], the controller is still a single logical entity. In the decentralized approach described in this thesis each controller is standalone and independent from the others. In distributed architectures the different controllers coordinate among them through the so called east-west interface. Via east-west communication, the different instances are tied together into a single logical controller. Contrary, the decentralized controllers do not exchange messages among themselves (i.e. there is no east-west interface), they only -indirectly- coordinate through the global state that they share.

1.5 Methodology: Research-Standardize-Implement

First the idea of decoupling state from control and then the development of decoupled-state architectures, are both the result of an iterative process that involved research, standardization and implementation. In this section we briefly describe the iterative methodology that has been applied in this thesis, which we also summarize in Fig. 1.5.

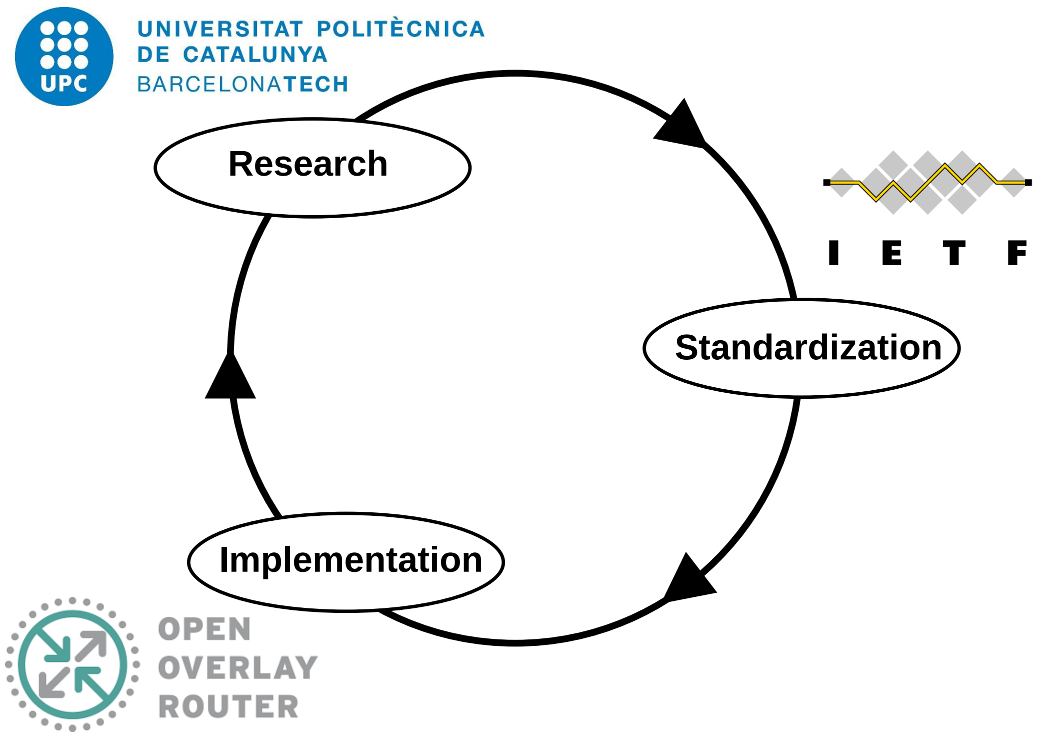


Figure 1.5 **Thesis methodology**

Research The research contained in this thesis has been performed at the academia, although due to its applied nature we have been in close contact with the industry. Therefore, the topics investigated on this thesis have been brought by industry needs and motivated by real shortcomings of existing deployments. Following that vision, the focus of this thesis has been to identify functionality gaps and space for improvement in current applied approaches. Nevertheless, there has been a high awareness of current industry status while investigating how to push the state of the art on those fronts. In that sense, we have tried to propose approaches interoperable with existing deployments and solutions feasible in the short or mid term.

Standardization In some cases, the results obtained through research can not be directly applied to the industry since there is a lack of proper networking protocols or features on those already existing. Therefore, new protocols or extensions needed to be defined before the research could be applied. In other cases, defining these protocols and/or extensions was the objective of the research in the first place. However, to make new protocols/extensions available and bring them to the industry they need to go through an standardization process. In this thesis we have extended different existing networking protocols and propose different mechanisms to enable new features in existing solutions. This standardization process has been mostly carried out at the Internet Engineering Task Force (IETF), the major venue for standardization of Internet related protocols. The standardization process entitled us to bring our tentative ideas to the community at large and receive feedback from networking experts. This allowed us to refine our designs and ensure that our proposals were feasible and interoperable with existing solutions.

Implementation Once required protocols or extensions are available as a result of the standardization process, it is possible to actually implement the solutions defined in the research phase. In this thesis we strongly leverage on validating our research via real implementation rather than using simulation or analytical tools. The architectures and discussions presented in this thesis are supported by solid prototyping and actual deployments. For that we have made a strong use of the OpenOverlayRouter (OOR) project [100]. OOR is a project maintained by our research group that enables to deploy programmable overlay networks and offers a flexible platform for rapid development and easy prototyping. During the span of this thesis several features arose as a result of the research conducted and were added to OOR. Besides our in-house OOR project, we have also leveraged on the OpenDaylight controller [103] to which we have also contributed. Finally, it should be noted the conclusions gather from the implementation and prototyping experience leaded to rethink the proposals conceived during the research phase or opened the path for new research topics.

1.6 Outline and Contributions of this Thesis

To summarize the scope and contributions of this thesis, in this section we briefly describe the outline of this document, the contents of each part and the contributions associated to each chapter.

1.6.1 Introduction

To put this work into context, we have given an historical overview of legacy networking and how it leaded to the need for SDN. After that we have summarized the origin and evolution of SDN and the different designs for SDN controllers over the years. From this analysis on the state-of-the-art on SDN we concluded that current controller approaches do not suffice to fulfill the requirements of certain scenarios. In that sense, we introduced the idea that it may be beneficial to explore taking the state out of the SDN controller. The concept of decoupling state and control in SDN is the motivation of this thesis and its main contribution.

1.6.2 Part I: Decoupled-State Architectures

In Part I, we introduce two different types of decoupled-state SDN architectures (asynchronous and decentralized) and propose instances of both types designed for specific use-cases. Interestingly, the different solutions that we propose leverage on the Locator/ID Separation Protocol (LISP) [30] as the protocol for state exchange. Therefore prior to delve into the details of the architectures we analyze LISP as a protocol for SDN.

In Chapter 2 we describe the use of LISP as an SDN protocol and analyze LISP architecture and components from the point of view of SDN. This leads to highlight the particularities of LISP and to present the benefits and drawbacks of using LISP for SDN. Furthermore, we build a prototype to validate the analysis and to show the feasibility of an SDN solution based on LISP. After analyzing LISP we present two different LISP-based decoupled-state SDN proposals and the scenarios that motivate them.

On one hand, asynchronous decoupled-state architectures are suitable when data-plane devices are subject to a high churn. This is the case for end-nodes, such smartphones and home-routers, since they are transient and/or highly mobile. Following this idea, Chapter 3 proposes a SDN architecture specifically designed for end-nodes. The architecture leverages on distributed and symmetric controller nodes that offer an intent-driven northbound to the control applications and on a state database provisioned with a connectionless pull-based southbound towards the data-plane nodes.

On the other hand, this work highlights the case of Network Function Virtualization (NFV) for operator networks as one case where the SDN centralization may be impractical. NFV for operators can benefit from a decoupled-state architecture that decentralizes the control and achieves better local processing. For this scenario Chapter 4 describes an architecture where the state remains centralized, but the controllers are pushed close to the data-plane devices.

Contributions of Part I

- Chapter 2 - LISP as a Protocol for State Exchange in SDN
  - **Paper**: Rodriguez-Natal, Alberto, Marc Portoles-Comeras, Vina Ermagan, Darrel Lewis, Dino Farinacci, Fabio Maino, and Albert Cabellos-Aparicio. "LISP: a southbound SDN protocol?" *Communications Magazine, IEEE* 53.7 (2015): 201-207.
  - **Code**: Original LISP northbound interface for the OpenDaylight SDN controller. https://github.com/opendaylight/lispflowmapping
  - **Internet-Draft** Ermagan, Vina, Alberto Rodriguez-Natal, Florin Coras, Albert Cabellos-Aparicio, and Fabio Maino. "LISP Configuration YANG Model", *draft-*

ietf-lisp-yang-01, December 2015, (work in progress).
https://tools.ietf.org/html/draft-ietf-lisp-yang-01

  - **Internet-Draft** Rodriguez-Natal, Alberto, Albert Cabellos-Aparicio, Vina Ermagan, Fabio Maino, and Sharon Barkai. "MS-originated SMRs", *draft-rodrigueznatal-lisp-ms-smr-01*, April 2016, (work in progress).
    https://tools.ietf.org/html/draft-rodrigueznatal-lisp-ms-smr-01
  - **Internet-Draft** Rodriguez-Natal, Alberto, Albert Cabellos-Aparicio, Sharon Barkai, Vina Ermagan, Darrel Lewis, Fabio Maino, and Dino Farinacci. "LISP support for Multi-Tuple EIDs", *draft-rodrigueznatal-lisp-multi-tuple-eids-01*, January 2016, (work in progress).
    https://tools.ietf.org/html/draft-rodrigueznatal-lisp-multi-tuple-eids-01

- Chapter 3 - Asynchronous SDN Architecture for End-Nodes
  - **Paper**: Rodriguez-Natal, Alberto, Vina Ermagan, Kien Nguyen, Sharon Barkai, Yusheng Ji, Fabio Maino, and Albert Cabellos-Aparicio. “SDN for End-Nodes.” (under review).

- Chapter 4 - Decentralized SDN Architecture for Operators NFV
  - **Paper**: Rodriguez-Natal, Alberto, Vina Ermagan, Ariel Noy, Ajay Sahai, Gidi Kaempfer, Sharon Barkai, Fabio Maino, and Albert Cabellos-Aparicio. “Global state, local decisions: Decentralized NFV for ISPs via enhanced SDN.” (under review).
  - **Internet-Draft**: Barkai, Sharon, Dino Farinacci, David Meyer, Fabio Maino, Vina Ermagan, Alberto Rodriguez-Natal, and Albert Cabellos-Aparicio. "LISP Based FlowMapping for Scaling NFV", *draft-barkai-lisp-nfv-07*, December 2015, (work in progress).
    https://tools.ietf.org/html/draft-barkai-lisp-nfv-07

1.6.3 Part II: Deploying SDN for End-Nodes

This thesis identifies SDN for end-nodes as an open-field for study and research. As it has been already mentioned, in Chapter 3 we explore an asynchronous architecture for end-nodes that relies on the LISP protocol for state exchange. In Part II we go deeper into the implications of making SDN available for end-nodes, and particularly of doing so via LISP. In that sense, in Chapter 5 we analyze the particularities of LISP when applied to mobility scenarios. Particularly, we discuss location and identity privacy issues of the LISP protocol when used for mobile-nodes.

Furthermore, in Chapter 6 we describe and benchmark OpenOverlayRouter, an open-source overlay software that implements LISP to enable SDN at end-nodes.

Contributions of Part II

- Chapter 5 - Privacy for LISP Mobile Nodes
  - **Paper**: Rodriguez-Natal, Alberto, Lorand Jakab, Marc Portoles-Comeras, Vina Ermagan, Preethi Natarajan, Fabio Maino, and Albert Cabellos-Aparicio. "LISP-MN: mobile networking through LISP." *Wireless personal communications* 70.1 (2013): 253-266.
  - **Paper**: Rodriguez-Natal, Alberto, Lorand Jakab, Vina Ermagan, Preethi Natarajan, Fabio Maino, and Albert Cabellos-Aparicio. "Location and identity privacy for LISP-MN." *International Conference on Communications (ICC), IEEE,* 2015.
- Chapter 6 - OpenOverlayRouter: Architecture and Evaluation
  - **Paper**: Rodriguez-Natal, Alberto, Florin Coras, Albert Lopez-Bresco, Lorand Jakab, Marc Portoles-Comeras, Preethi Natarajan, Vina Ermagan, David Meyer, Dino Farinacci, Fabio Maino, and Albert Cabellos-Aparicio. “OpenOverlayRouter: Architecture and Performance.” (under review).
  - **Code**: Developer and maintainer for the OpenOverlayRouter project (formerly LISPmob), an open-source implementation to deploy programmable overlays. http://www.openoverlayrouter.org/

1.6.4 Thesis Summary and Open Research

Finally, this work explores the implications and future work regarding decoupled-state SDN architectures in Chapter 7. Particularly, we expose that the interfaces to interact with a decoupled-state require further investigation and that an standalone state database requires of careful analysis.

Part I

Decoupled State Architectures

Chapter 2

LISP as a Protocol for State Exchange in SDN

This chapter describes how the Locator/ID Separation Protocol (LISP) can be used as a protocol for SDN. LISP basic idea is to split current IP addresses overlapping semantics of identity and location in two separate namespaces. Since its inception the protocol has gained considerable attention from both the industry and the 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. LISP is specially interesting as an SDN protocol since it can fit as a state exchange protocol for decoupled-state architectures. In this chapter we try to fill the gap and present 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 OpenOverlayRouter open-source implementation of LISP. Finally, it should be noted that although Chapter 3 and Chapter 4 show the benefits of using LISP for decoupled-state SDN architectures, this chapter present LISP as a general southbound protocol for SDN, and thus not constrains its usage for decoupled-state architectures only.

2.1 Introduction

The LISP [30] decouples identity from location on IP addresses by creating two separate namespaces, EIDs (Endpoint Identifiers) to identify hosts and RLOCs (Routing Locators) to route packets over transit networks. LISP follows a map-and-encap approach to map identifiers to locators and encapsulate EID traffic into RLOC packets. EID packets are intercepted by border routers and encapsulated towards the RLOC of the router serving the destination EID domain. On the other hand, the LISP control-plane uses a publicly accessible distributed database -known as Mapping System- to register and request EID-to-RLOC mappings. With this LISP effectively decouples control and data planes and creates a dynamic EID-based overlay network on top of an RLOC-based underlay, that can be *programmed* by simply interacting with the Mapping System.

LISP original purpose 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 [80] that includes more than 20 countries and hundreds of institutions. LISP hardware and software are also widely available, both in open-source [100, 104] and proprietary implementations [81].

As a result of the LISP standardization and research efforts, the protocol has grown architecturally and has been applied to use cases beyond its original purpose. Since its inception, LISP has gained a significant traction on both the industry and the academia due to the possibilities of its programmable overlays [129]. LISP-enabled dynamic overlays provide a standardized, inter-domain, and programmable framework to enable low-CAPEX innovation at the network layer. Therefore, there is a growing interest on the role of LISP in Software-Defined Networking (SDN) and on how SDN can benefit from LISP overlays. Interestingly, LISP is already becoming part of SDN solutions such the OpenDaylight controller [103]. In this chapter we analyze the relation between the LISP architecture and the SDN paradigm.

As described in Chapter 1, 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 in defining the high level abstraction interface (see Frenetic [36] or Procera [66] as examples). With respect to the southbound interface there are as well several options, being OpenFlow [87] the one that has gained major widespread on the industry.

One of the contribution of this thesis is to analyze LISP as a southbound SDN protocol. For this, this chapter presents a systematic analysis of the fundamental SDN requirements -inferred from the literature [61, 36, 66, 87, 131, 76, 144, 13, 71]- 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 OpenOverlayRouter (formerly LISPmob) open-source implementation.

2.2 Background: Locator/ID Separation Protocol (LISP)

Current IP addresses convey the semantics of both identity and location, which has been considered an issue since the early days of networking [133]. Trying to alleviate this semantic overload LISP decouples host identity from its location by creating two different namespaces: Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). Hosts are identified by an EID, and topological network locations by RLOCs. 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 addresses) to be used.

Packets are routed based on EIDs at LISP sites and on RLOCs on transit networks. LISP follows a map-and-encap approach to allow transit between EID and RLOC space. EID identifiers are mapped to RLOC addresses and EID traffic is encapsulated into RLOC packets. At the edge points of LISP sites Ingress/Egress Tunnel Routers (xTR) are deployed to serve as gateway. In the case of a packet willing to leave the LISP site, the xTR performs a lookup at its map-cache to know how to encapsulate the EID packet into an RLOC packet. The map-cache contains the mappings from EID to RLOC that the xTR has learned. If there is already an entry, the xTR can obtain the RLOC that corresponds to the destination EID. Then, it encapsulates the original packet within a new IP packet, using this RLOC as destination and its own RLOC as source, and forwards the packet to the transit network. If there is no entry for the destination EID, the xTR needs to find first the EID-to-RLOC mapping via the Mapping System

The Mapping System is a distributed database containing EID-to-RLOC mappings. A single EID can be mapped to several RLOCs with different priority and weight values to allow backup policies and traffic balancing. The Mapping System is composed by Map-Resolvers and Map-Servers. In short, Map-Servers store mapping information and Map-Resolvers find the Map-Server storing a specific mapping. There are several approaches for the internal architecture of the Mapping System, i.e. how a Map-Resolver locates the appropriate Map-Server, as example see [55, 60]. To populate the Mapping System, xTRs register in the Mapping System the mappings they are in charge of (i.e. their local RLOC addresses and the EID space of their LISP site) via Map-Register messages. Depending on the deployment scenario, it is also possible that Map Servers can be proactively provisioned with mappings without requiring xTRs to send Map-Register messages.

These mappings are requested by the xTRs, when they have a map-cache miss, via a Map-Request message sent to a Map Resolver. The Map-Request is routed internally within the

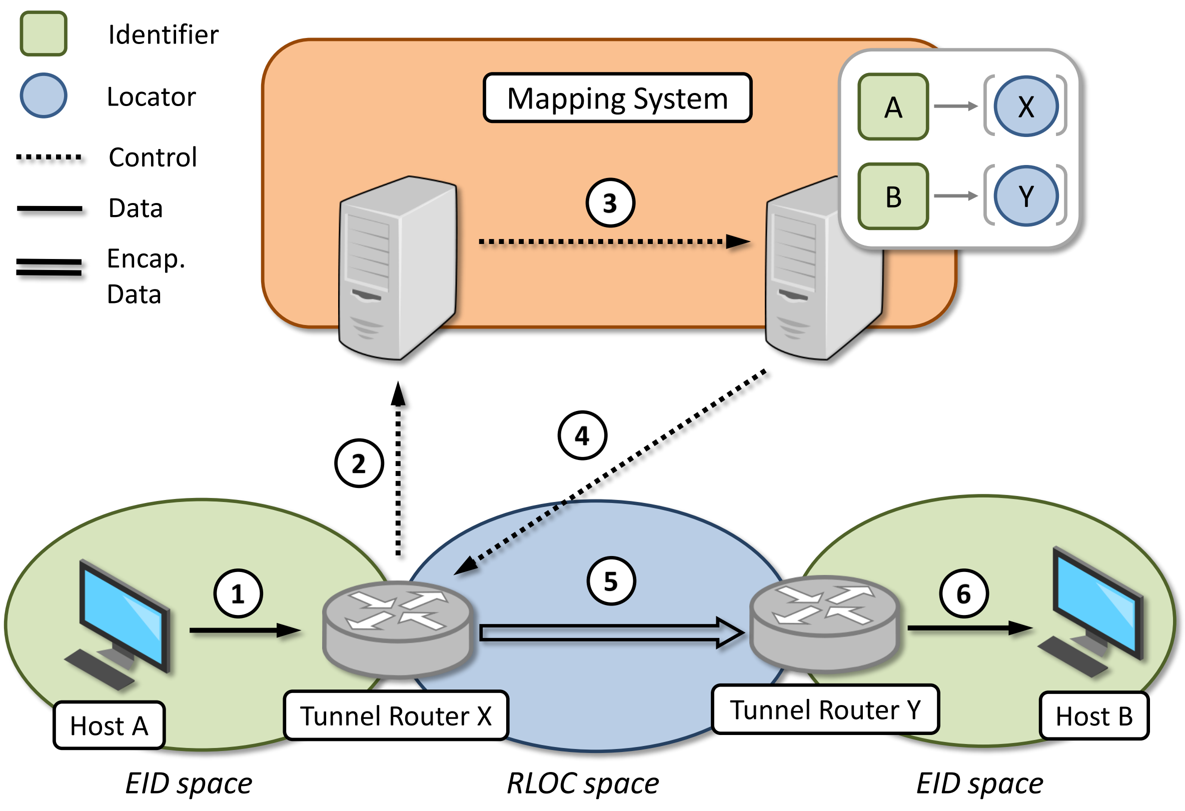


Figure 2.1 **LISP Overview**

Mapping System until it is received by the Map-Server storing the mapping. The Map-Server then sends the requested mapping to the requester xTR via a Map-Reply message (in some cases, if the mapping was registered by an xTR the Map-Server may instead forward the Map-Request to it). The retrieved mapping is stored in the xTR's map-cache for future use.

If there is no mapping information stored on the Mapping System for the EID requested, the xTR will receive an empty Map-Reply. That way the xTR knows the destination is not in the LISP space. However, in some cases it can encapsulate the traffic towards a Proxy xTR (PxTR). The PxTR will decapsulate the LISP packets and forward the regular traffic towards the legacy Internet. Furthermore, it is possible to enforce path policies for encapsulated packets by means of deploying Re-Encapsulation Tunnel Routers (RTRs) on the transit network. LISP encapsulated traffic can be forwarded to an RTR instead of to its destination xTR. At the RTR the traffic will be then re-encapsulated towards a next hop (either its destination xTR or another RTR). Finally, end-nodes may be provisioned directly with LISP capabilities if they implement the LISP Mobile Node (LISP-MN) specification [32, 125] described in Section 5.2.

Figure 2.1 depicts 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 (e.g. the Internet). To allow transit between EID and RLOC space, a map-and-encap approach is performed by the LISP Tunnel Router X (i.e. xTR X) deployed at the edge point. 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 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 towards Tunnel Router Y ⑤. Upon arrival at destination, Tunnel Router Y will decapsulate the packets and forward them natively to host B ⑥.

2.3 LISP: An SDN Architecture?

In this section we analyze if the LISP architecture -as is- 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 [87] 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 router level, implemented on the Tunnel Routers, all control is pushed to the Mapping System.

Network programmability Frenetic [36] and Procera [66] are two examples of the interest of the community on 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 ([36, 66])

and focuses on representing network state, therefore LISP should be complemented by a rich northbound language.

Centralized control *Levin, et al.* [76] expose 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 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.* [144] show the concern of the SDN community about SDN scalability. LISP is a pull-based architecture that stores the state in the Mapping System, network entities (e.g, LISP Tunnel Routers) retrieve and cache only locally relevant state on demand. The literature shows that the Mapping System internals can be designed to be scalable [60]. Furthermore, the connectionless approach used by LISP reduces the signaling overhead and the burden on the controller to establish and maintain sessions.

Core-Edge split *Casado et al.* [13] analyze the main shortcomings of existing SDN architectures and point 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 switches function, 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.

2.4 Architectural Analysis

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.

2.4.1 Flexible Namespace

The main LISP specification assumes IPv4 and IPv6 as address families, however it is flexible enough to allow using any other address families (for instance MAC addresses). 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 addresses 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, etc. 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

2.4.2 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 figure 2.2. 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 [71] does with its own distributed databases.

Onix is a well-known wide-area SDN deployment, it 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.

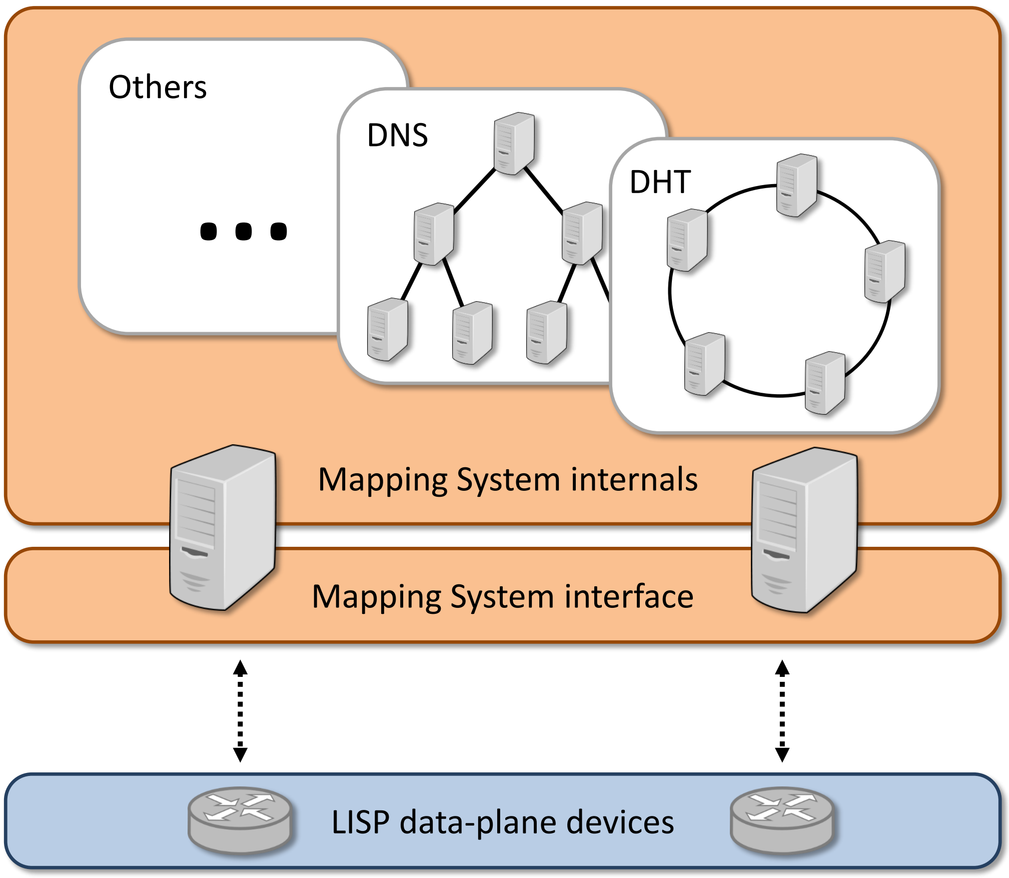


Figure 2.2 **LISP Mapping System**

With a LISP deployment, Onix (or any later Onix-based architecture) 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 and 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 2.2 also shows how the Mapping System can use different internal implementations.

A Mapping System indexing common IP addresses benefits from a hierarchical structure, such DNS. This is the approach followed by the Delegated Database Tree (DDT) based on [60], 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 one, should be used. Although some initial efforts towards a DHT-like Mapping System can be found in the literature [85], at the time of this writing only a hierarchical Mapping System (DDT) has been successfully widely deployed.

Consistency *Levin, et al.* [76] 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. 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 instantaneous reflected on the data-plane. For instance a LISP Tunnel Router can register new mapping information into the Mapping System at any time, however 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 [57] and second, a special control message can be used to explicitly notify remote parties of the mapping update [30].

2.4.3 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-encapsulate 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 like firewalls, traffic analyzers, accounting points, can be plugged, implemented or virtualized on top of re-encapsulating devices. In that sense, figure 2.3 shows the abstract representation of a Re-Encapsulating Tunnel Router device with some network functions integrated that are used on demand.

2.4.4 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

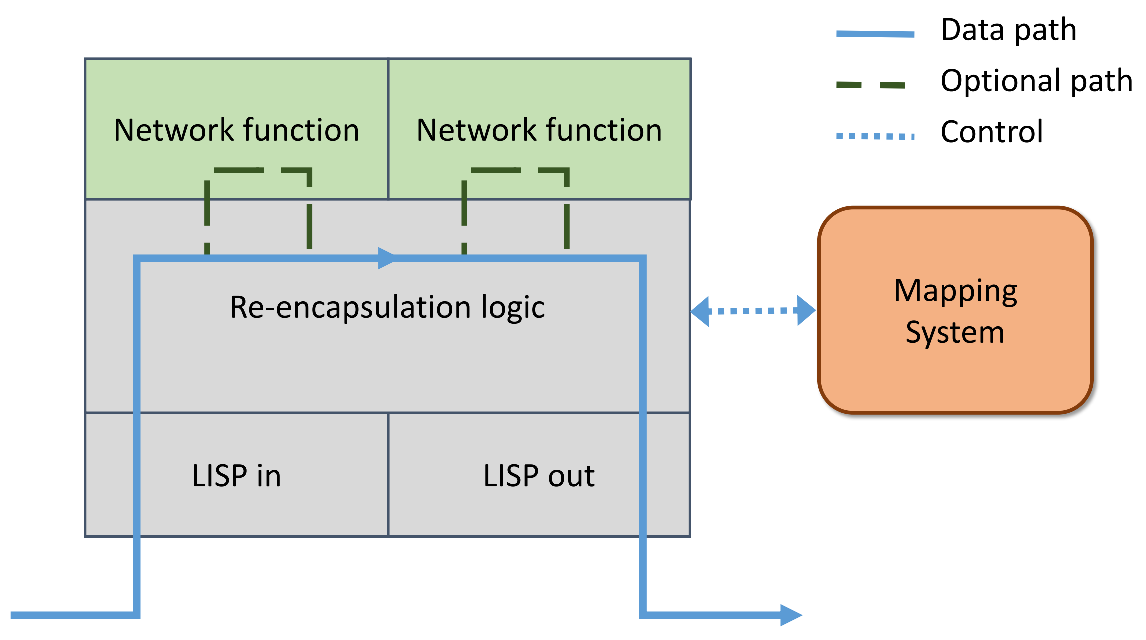


Figure 2.3 **LISP Re-Encapsulating Tunnel Router**

of the Explicit Locator Path (ELP). An Explicit Locator Path is a list of hops through where 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 locators 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 2.4 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.

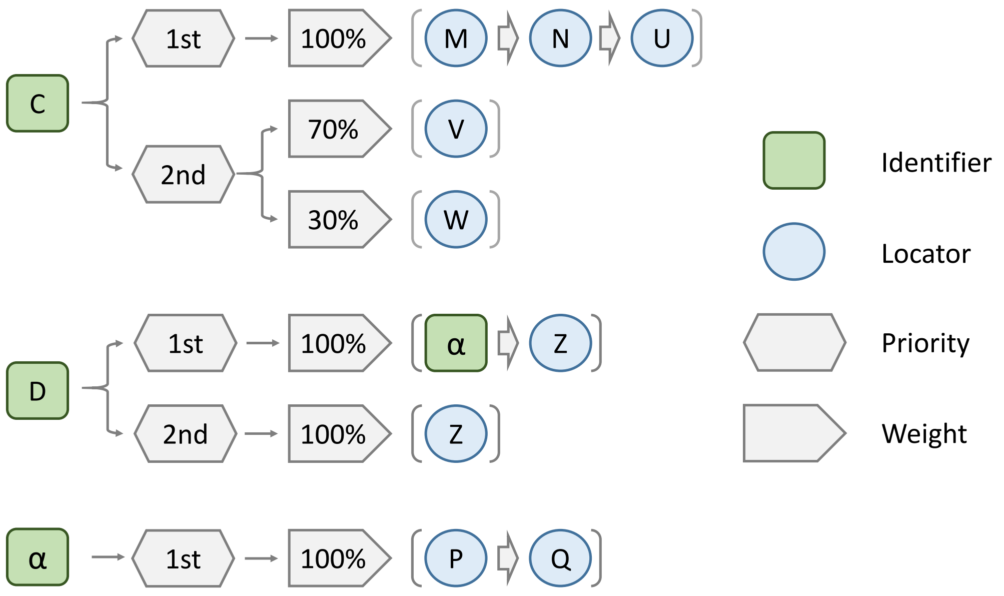


Figure 2.4 **LISP Traffic Engineering**

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 enables network programmability due to the ability to define custom programmable paths for packets on 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.

2.4.5 Label system

Instance-ID is a 24 bit length identifier that can be associated to 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 like Fabric [13], 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.

2.5 Discussion

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

2.5.1 Highlights

Based on the analysis of Section 2.3 and Section 2.4 we highlight the most relevant features of LISP in SDN environments.

- *Scalability*: As described on Section 2.3 and Section 2.4.2, the Mapping System and LISP connectionless pull-based signaling eases the scalability of LISP system. An SDN solution can use LISP to provide a scalable network state database that can be directly queried by both data and control devices
- *Interoperability*: Given its flexible namespace (Section 2.4.1) and its label system (Section 2.4.5), LISP is agnostic to the protocols it encapsulates and is well-suited to deploy cross-technology overlays.
- *Inter-domain*: Network landmarks (Section 2.4.3) and LISP traffic engineering capabilities (Section 2.4.4) allow LISP to enforce policies on transit networks and make it suitable for inter-domain deployments.

2.5.2 Benefits

First LISP has been designed to be incrementally deployable and to leverage on 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 [140]. 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 to 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.

2.5.3 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. This can introduce latency in the system and/or produce packet losses.
- *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 to the LISP architecture, however they do not have a strong impact on performance given that first, LISP encapsulation typically adds only 36 bytes (IPv4) or 56 bytes (IPv6) [30], and second the LISP entities cache the mappings and because of the strong locality of traffic [16] achieve a hit-rate above 99%.

Regarding *Flat data support*, the limitation can be solved with a DHT-based Mapping System [85]. 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 potential enhancements for the protocol in Section 2.5.4.

2.5.4 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 [99] and standardized at the IETF [123]. 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 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 pre-defined 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 [122], despite in the future further look-up processes can be implemented, potentially leveraging on OpenFlow tuple matching process.

2.6 Prototype

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

2.6.1 Setup

The prototype topology is depicted in figure 2.5. 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 OpenOverlayRouter (OOR) implementation (formerly LISPmob), that implements both LISP control and data planes. We slightly modified OOR to support look-ups based on destination-source EID tuples.

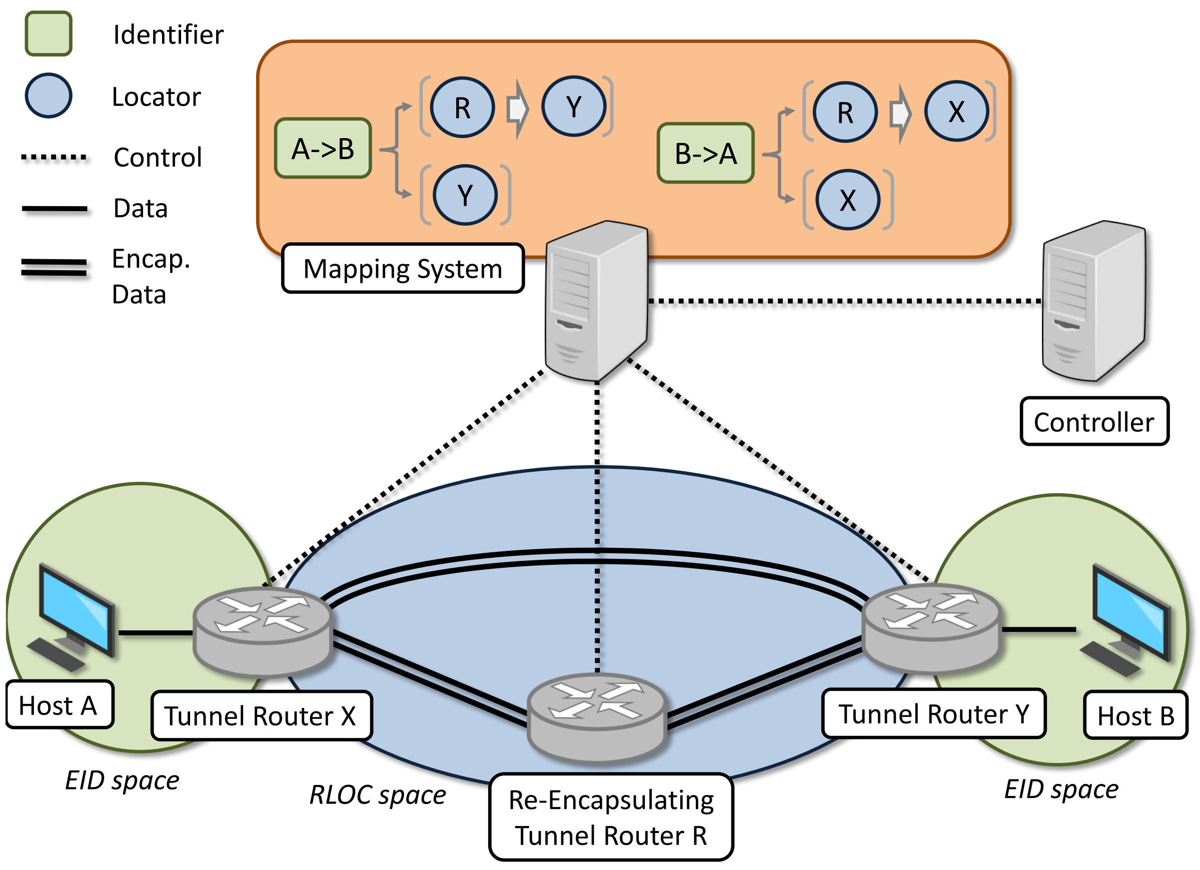


Figure 2.5 **LISP SDN prototype**

On the described prototype we test two different scenarios, one where the traffic goes directly to its destination and another one 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 more priority.

2.6.2 Metrics

To extract relevant metrics we run 10 iterations injecting ping packets 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 (see Section 2.5.3) when we send a flow over a new path. OOR does not put any packet on hold on this scenario, therefore the packets are dropped until the mapping has been resolved. 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.

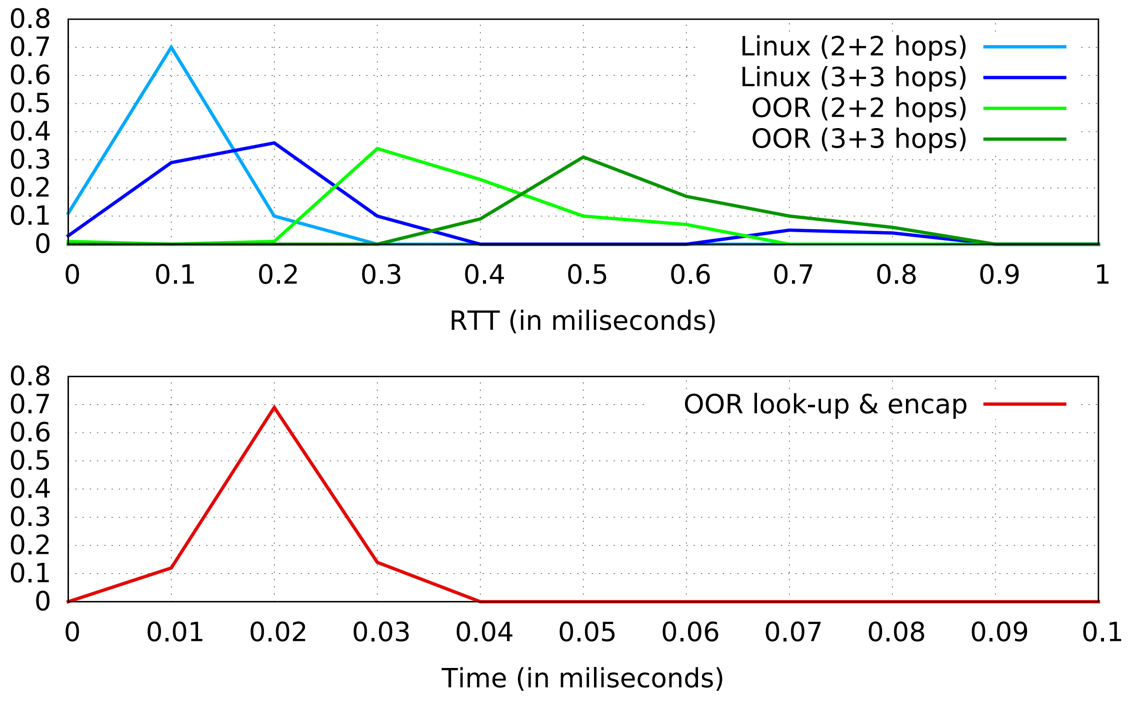


Figure 2.6 **OOR induced delay**

Delay To measure how much delay is introduced by OOR we built an equivalent prototype without LISP capabilities, where traffic paths were configured modifying the routing tables on the Linux boxes. The top of figure 2.6 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 4 hops (i.e. 2 from A to B and 2 from B to A), while the detour introduces one extra hop in each direction (3+3) for a total of 6 hops. The bottom part of figure 2.6 shows how much time elapses since OOR receives a new packet until it delivers the LISP encapsulated packet, i.e. LISP look-up and encapsulation. The plots in figure 2.6 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 $\leftrightarrow$ kernel communication required by OOR. Nevertheless, the lookup and encapsulation operations may be optimized by router manufactures to enable the performance of hardware implementations to be similar to that of traditional IP datagram forwarding.

2.7 Conclusions

In this chapter we have analyzed if LISP -as is- 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 specially suitable for SDN deployments over legacy or transit networks, such 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 advanced look-up process. Finally, the presented prototype demonstrates that LISP is feasible for SDN scenarios.

Chapter 3

Asynchronous SDN Architecture for End-Nodes

The advent of SDN has brought a plethora of new architectures and controller designs for many use-cases and scenarios. Existing SDN deployments focus on campus, datacenter and WAN networks. However, little research efforts have been devoted to the scenario of effectively controlling a full deployment of end-nodes (e.g. mobile devices, home routers, etc.) that are transient and scattered across the Internet. In this chapter we present a rigorous analysis of the challenges associated with an SDN architecture for end-nodes, show that such challenges are not found in existing SDN scenarios, and provide practical design guidelines to address them. Then, and following these guidelines we present an decoupled-state architecture specifically designed for the end-nodes scenario. This architecture is based on the combination of a distributed controller with an intent-driven northbound and a decoupled Network Information Base provisioned with a connectionless pull-oriented southbound. Finally, we measure a proof-of-concept deployment to assess the validity of the analysis as well as the architecture.

3.1 Introduction

In the recent years the Software-Defined Networking (SDN) [74] paradigm has been applied to different scenarios, namely campus, datacenter and WAN networks [74, 71, 5]. However, little research efforts have been devoted to scenarios where the controlees are end-nodes, e.g. mobile nodes (smartphones, tablets, etc), personal computers, home or SOHO (Small Office or Home Office) routers, etc. Despite the scarce literature on the topic, we believe that an SDN architecture designed for end-nodes enables interesting use-cases, since companies and operators can push policies and control the traffic from its very origin.

However, this novel SDN scenario brings a set of challenges that are not found in existing SDN deployments or that are strongly exacerbated in the end-nodes case. On one hand, the number of controlees is very large and they are typically scattered through various networks, this imposes very high scalability and availability requirements to the controller. On the other hand, this scenario comes with a high churn, since end-nodes are transient and mobile and might connect and disconnect randomly [29]. Finally, the architecture has to face an additional complexity burden to operate the data-plane. As opposed to transit routers and switches, end-nodes do not aggregate traffic and thus, require fine-grain policies.

In this chapter we analyze the challenges of deploying SDN for end-nodes and discuss practical design guidelines. As a result, we present a specific architecture based on a decentralized and symmetric controller with an intent-driven northbound and a pull-based southbound. We discuss the implications of the architecture and provide a proof-of-concept prototype that shows the validity of the analysis and the feasibility of the architecture. To conclude the chapter we provide a quantitative comparison of the proposed architecture with relevant related work.

3.2 Use-Cases

Network administrators can enhance the effective control that they have over end-nodes by provisioning them with SDN-capabilities. This entitles to extend the fine-grain control and centralized management enabled by SDN to the very end of the network. We can differentiate three different use-cases categories.

Source control SDN capabilities at end-nodes enable dynamic and remote control of their interfaces, which enables centralized management for load balancing and bandwidth aggregation. An operator would be able to balance a smartphone's traffic between Wi-Fi and cellular interfaces based on business agreements, remaining battery and/or device's geo-location (e.g. avoid traffic over cellular network when abroad). A remote backup company can leverage on multi-homed end-nodes to aggregate the bandwidth of all available interfaces and speed up transmissions.

Destination control SDN for end-nodes enables simpler and fine-grain control on the traffic destination. For instance, a content provider can detour traffic to one datacenter or another based on central control per end-node basis (e.g. to offer personalized content per end-user). Similarly, another company can offer remote VPN (Virtual Private Network) management on the devices of its employees dynamically tunneling the traffic depending on the destination (e.g. to decide on-the-fly which traffic should be encrypted).

Path control : Path control can be achieved by combining SDN-capable end-nodes with some well-placed SDN nodes on transit networks. This allows companies to steer traffic through transit SDN nodes to, for instance, optimize routing or apply in-path functions. As an example, a parental filtering company can make use of such capabilities to filter traffic going to the end-nodes and block inappropriate content.

3.3 Challenges

Achieving SDN support at the end-nodes is not straightforward since it presents a set of novel challenges when compared to existing SDN deployments. First of all, there is a *high number* of controlees, potentially reaching hundreds of thousands of nodes. Although existing SDN deployments may handle a large number of controlees, this burden is heightened in the end-node scenario when combined with the rest of its challenges.

In addition, in existing SDN deployments controlees are physically connected to the SDN network, however in the end-nodes scenario the controlees are attached to *legacy networks* at the edge. Such networks are usually out of the administrative control of the SDN domain. Moreover, the controlees are *scattered* over these edge networks, potentially at a worldwide scale.

Furthermore, the discussed scenario suffers from a high churn since end-nodes are *transient* and highly mobile, this makes a difference with regular SDN where typically controlees are stationary and always available. Besides, and contrary to common SDN devices, end-nodes are subject to high *heterogeneity* in terms of capabilities and conditions (e.g. geo-location, spectrum usage, remaining battery, etc). Finally, end-nodes have to face *lower traffic locality* in contrast to typical SDN devices that usually aggregate traffic and take advantage of this to speedup data-plane performance using caches, wildcards and/or longest prefix matches.

3.4 Design Guidelines

The resulting design principles to overcome the scenario challenges are summarized in Table 3.1. First, an *overlay approach* that encapsulates traffic at the data-plane is required to bypass in-place networks and to offer a homogeneous view of the heterogeneous controlees. To be able to encapsulate overlay traffic into underlay packets, the end-node has to retrieve the appropriate rules by querying the NIB (Network Information Base), term that we borrow from [71]. Due to the large number of data-plane nodes and the low traffic locality, this generates a high ratio of requests from the southbound to the NIB. Therefore, the controller must be able to *scale-out* to accommodate this very high volume of messages, while remaining *decentralized and*

In-place networks	High # of controlees	Scattered nodes	Low traffic locality	Transient devices	Heterogeneity	
●					●	Overlay deployment
	●		●			Scale-out architecture
		●		●		Decentralized/symmetric
	●		●	●		Connectionless pull-based
	●			●	●	Intent-driven
			●			IP granularity

Table 3.1 **Challenges to design principles**

symmetric since end-nodes are scattered and mobile and the queries may reach any controller node. Considering these requirements we propose to use as NIB back-end a distributed database designed for availability and partition tolerance, even if it offers eventual consistency.

Generally in SDN, data-plane nodes establish a connection with the controller to exchange state, however end-nodes are transient and mobile, thus a regular connection-based southbound protocol would impose frequent connection reestablishment. As a consequence an important design guideline is to use a *connectionless southbound* that mitigates this burden and takes advantage of the symmetry of the controller to decouple controlees from specific controller nodes. This eases end-nodes roaming and also simplifies on-demand controller nodes instantiation (i.e. a new controller node can transparently start serving end-nodes previously served by another). On the other hand, the controller can not push the state to transient nodes, since they might be not available at the time the state is generated. To address this constrain, end-nodes must use a *pull-based mechanism* for the southbound that allow them to retrieve the state on demand.

In addition, as opposed to transit routers and switches, end-nodes do not aggregate traffic, which results in low traffic locality and a poor performance of caching techniques. In this context it is not advisable to follow the common multi-tuple lookup granularity used in SDN deployments, and rather it is recommended to use just *IP granularity*. This simplifies data-plane implementation and reduces the complexity at the NIB. We have also carefully reviewed all the use-cases discussed in Section 3.2 and found that IP granularity is enough to implement them. It is also worth noting that if a new use-case requires per-flow granularity, then the NIB data-model has to be carefully designed to trade-off the extra complexity (and associated increased latency) with the requirements of the new use-case. Additionally, it has to be considered the added flow setup time at the controlees against the latency requirements of the use-case.

Furthermore, the very large number of nodes to control, as well as the fact that they are mobile and transient, makes programming via imperative northbound interfaces very complex. In addition, the heterogeneity of the devices and their network conditions make challenging defining per end-node policies. As a result we suggest to use an *intent-driven approach* (i.e. a declarative language [36, 66, 93]) as the northbound interface, to define abstract policies to apply to data-plane traffic.

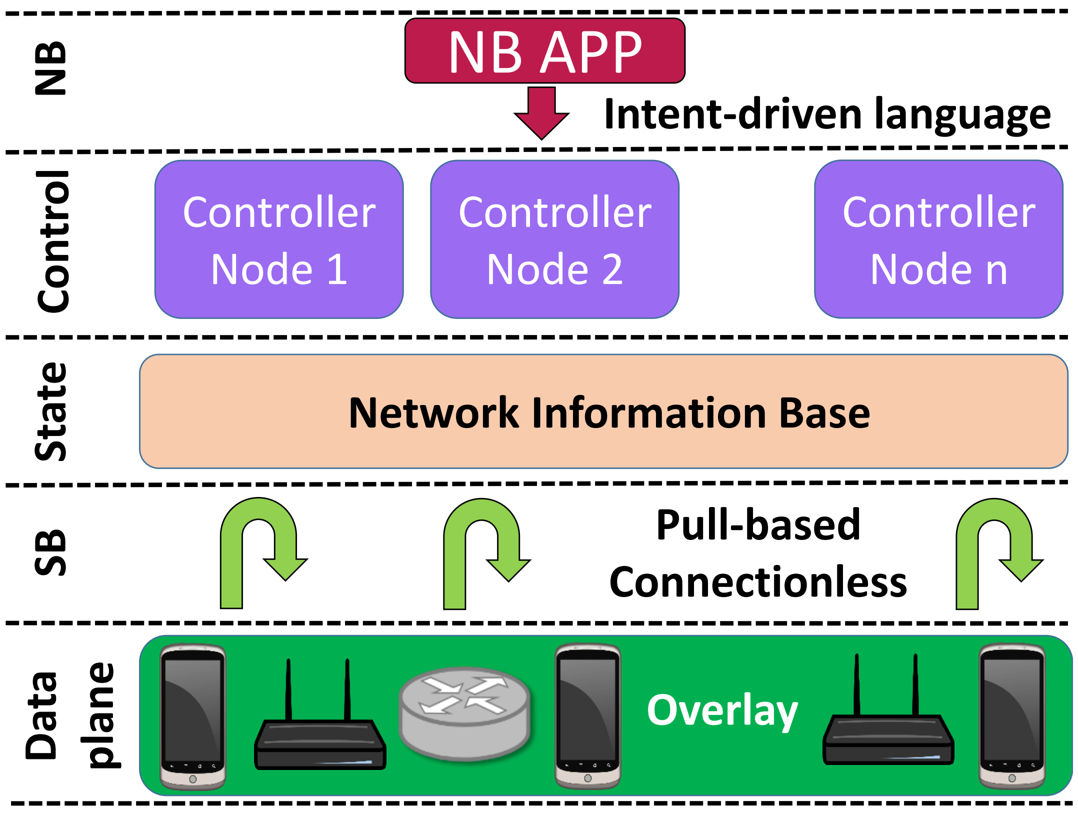


Figure 3.1 **Architecture**

3.5 An SDN Architecture for End-Nodes

Following the design principles from Section 3.4, we propose the decoupled-state architecture depicted in Fig. 3.1. The architecture centralizes all the state in the NIB, composed by different NIB nodes storing the partitions of a Cassandra database [75]. These NIB nodes are accessed by distributed and symmetric controller nodes. To simplify the description of the architecture, we consider that the physical machines hosting controller nodes also host NIB nodes (i.e. Cassandra partitions plus state exchange logic), although decoupled-state architecture does not impose such requirement.

The controller nodes offer an intent-driven northbound based on the GBP (Group Based Policy) [41] project to the control applications. The NIB nodes offer a connectionless pull-based southbound implemented via LISP (Locator/ID Separation Protocol) [30, 126] to the data-plane nodes. The data-plane nodes are provisioned with encapsulation capabilities to join a common overlay.

3.5.1 Northbound

The policies are enforced from the northbound using a declarative intent-driven language that is rendered at the controller to generate the data-plane state stored in the NIB. In particular we propose to use a group-oriented declarative language given that related nodes will likely share the same policies. To ease the implementation of the architecture we use the OpenStack [106] backed GBP project as the northbound language since we can leverage on its already defined syntax. Although, depending on the specific use-case, its syntax may need to be extended to support further primitives (e.g. geo-location).

In the GBP language, end-nodes belong to groups and policies define how to prioritize links, balance traffic or chain hops on the underlay. The policies can be defined for a pair of groups, when one sends traffic to the other, or in general for the group's default ingress and egress policies, thus avoiding defining all possible group pairs. For instance, a particular use-case can define the groups *Employees' Laptops* and *West Coast Office* with the following default policies, *Employees' laptops* should use *Ethernet* for egress traffic and *West Coast Office* ingress traffic should be balanced between *Link-1* and *Link-2*. However, the use-case may also define that when the traffic from *Employees' Laptops* goes to *West Coast Office* it should use *LTE* as egress, pass trough hop *Firewall* and use *Link-3* as ingress. More complex groups are possible, such *Employees' Laptops with battery below 15%*, however complex use-cases require carefully design, see Section 3.6.1.

3.5.2 Controller

The controller is supported by distributed, decentralized and symmetric nodes as the one depicted in Fig. 3.2. They have an intent parser module that processes the northbound polices that result in NIB state. The bootstrap module is used to initialize via NETCONF (Network Configuration Protocol) [24] data-plane nodes.

The controller nodes receive abstract policies from the northbound and render them into specific state to be stored at the NIB nodes. All the controller nodes have a connection to a Cassandra database where the NIB is stored. The controller nodes coordinate among themselves using a ZooKeeper [56] subsystem.

System Scalability

The architecture is able to scale-out using the Cassandra backend and symmetric design for both the controller and NIB nodes. On one hand, the NIB is stored in Cassandra and thus the system state will scale as good as Cassandra does. On the other hand, the controller and NIB nodes have been designed to be analogous and to be used indifferently, and since they are all connected to the same Cassandra back-end, they can all access the same state.

Since all the controller and NIB nodes are interchangeable and offer the same functionality, the system can be scaled-out smoothly and presents no single point of failure. If the number of controlees or the ratio of northbound operations increases, the controller can automatically coordinate (via the ZooKeeper subsystem) the instantiation of a new NIB node or a new controller node to redistribute the load.

This process is transparent to the data-plane nodes. From the southbound point of view, data-plane nodes will detect an overloaded (or down) NIB node by the lack of acknowledgment of their southbound requests and will eventually switch to a different one. This automatically balances the load across different nodes. The distributed and symmetric schema for the NIB nodes also encourages data-plane nodes to communicate with the NIB node that offers less latency and to seamlessly move to a different one if needed.

It is also interesting to point that the state cached in data-plane nodes, and the ephemeral state at the NIB (e.g. the list data-plane nodes registered) will eventually expire and be automatically removed. This solves the problem of unnecessary state stored that can burden the system performance and/or increase query time.

3.5.3 Network Information Base (NIB)

Each NIB node stores a Cassandra partition and is provisioned with a LISP interface, as shown in Fig. 3.2. Among the available NoSQL databases we choose Cassandra [75] as the back-end for the NIB since it is a general purpose database that offers huge scalability and almost constant query latency time regardless of the size of the stored state [119], see Section 3.6.2. Via a LISP interface each NIB node can exchange state with the southbound nodes.

Information Tables

For the scenario considered, the NIB does not need to represent a complete view of the network topology, just the relations and interactions among the groups and the nodes belonging to them. To that end, the NIB stored in Cassandra comprises three main tables, *nodes*, *groups* and *paths*. Queries to the NIB will result (through one or more database queries) into rendered state specific to the requester node (see Section 3.6.1).

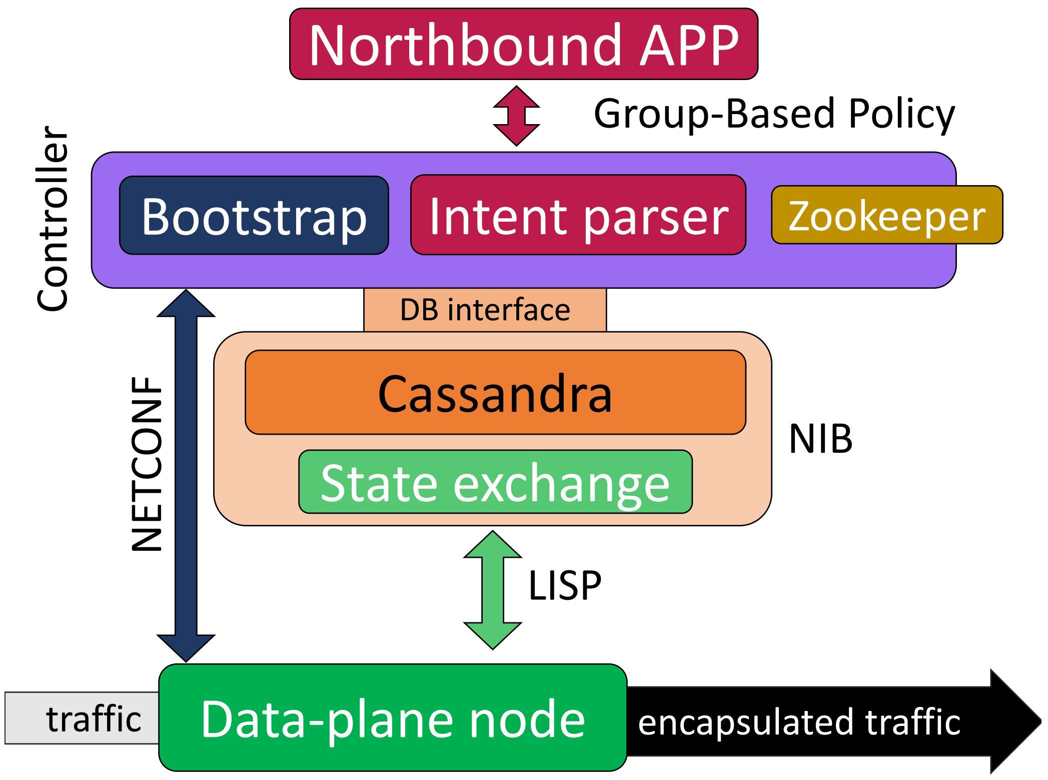


Figure 3.2 **Controller and NIB nodes detail**

Both *groups* and *nodes* provide ingress and egress policies pear each group/node. The *paths* table includes policies to apply only to traffic between a specific group pair. While for *groups* and *paths* the policies are in abstract form (e.g. ethernet preferred over Wi-Fi), in *nodes* they are rendered into specific network state per node. All the tables include a timeout-evicted list of requesters per entry, this is used to notify state changes if needed. The *groups* table also stores per each group, the current active nodes and group proxies (see Sections 3.5.4 and 3.5.5). The *nodes* table contains, besides the per-node-rendered policies, the node status (e.g. geo-location, remaining battery, last time seen). Finally, other auxiliary tables store information about the controller and NIB nodes themselves (underlay address, hardware specifications, recent load, etc). An example of the information that could be stored in the different NIB tables is provided below.

- *Data-plane nodes*: Node name (e.g. "Prof Alice's phone"), overlay identifier (e.g intranet IP address), node group (e.g. "Faculty phones"), general ingress/egress policies (e.g. prioritize WLAN over LTE), current points of attachment (e.g. current public IP of the WLAN iface), device state and status (e.g. geo-location, remaining battery), last time online (e.g. Unix timestamp), proxy nodes to use (e.g. "North Campus" proxy), network locators of requesters of their ingress data (e.g. a list of public IP addresses).

- *Groups*: Group name (e.g. "Faculty phones"), overlay identifier (e.g. intranet IP prefix), general policies per group (e.g. prioritize WLAN over cellular networks), policies to connect to other groups (e.g. when connecting to "Office network" use LTE), proxy nodes to use by its members (e.g. "North Campus" proxy), data-plane nodes belonging to the group (e.g. "Prof Alice's phone", "Ass. Prof. Bob's phone"), current active data-plane nodes (e.g. "Prof Alice's phone").
- *Paths*: Path name (e.g. "Secure access to office"), group-pair connected by the path (e.g. "Faculty phones" to "Office network"), overlay identifier (e.g. phones prefix to office prefix tuple), path-specific policies (e.g. overwrite interface priorities and prioritize LTE), path logical hops (e.g. "CS building" gateway $\rightarrow$ "Firewall" $\rightarrow$ "Office network" gateway), network locators of requesters of the path (e.g. a list of public IP addresses).
- *Controller/NIB nodes*: Location (e.g. public IP), hardware and network specifications (e.g. memory size, link throughput), average recent load (e.g. low).

State consistency

To keep the state updated at the NIB, data-plane nodes send periodical keep-alive LISP control messages registering their current status (e.g. battery, geo-location) at the NIB. These control messages can also be triggered as a response to a data-plane event. When there is a state update from a data-plane node (e.g. an interface goes down) the end-node sends an update to the NIB and notifies other interested data-plane nodes of the change (e.g. peers with which it is communicating).

When the state update comes from a northbound application (e.g. a modified group policy changes the interface priorities for a set of nodes) the controller renders the new state and installs it at the NIB. The insertion of new state may activate some database mechanisms that trigger state consistency callback operations (see [136] for details). Using this, the NIB notifies via the LISP protocol the state change to the data-plane nodes affected by it. If the change modifies state that other data-plane nodes may have cached, the NIB notifies those too (e.g. a node new ingress policy is notified to its requesters). Besides, the NIB has also to verify if any new state introduced by the northbound (e.g. creation of a path policy for a group-pair) should overwrite state previously cached at data-plane nodes (e.g. general destination-based cached state), in that case the NIB notifies the relevant parties of the insertion of this new state.

3.5.4 Southbound

The NIB state is pulled on demand by southbound nodes via LISP, a protocol to map overlay identifiers to underlay network locators. LISP expresses policies on the overlay as priorities and weights for the locators in the underlay. We advocate for LISP as the southbound protocol since it is connectionless, pull-based and overlay oriented [126].

State Retrieval

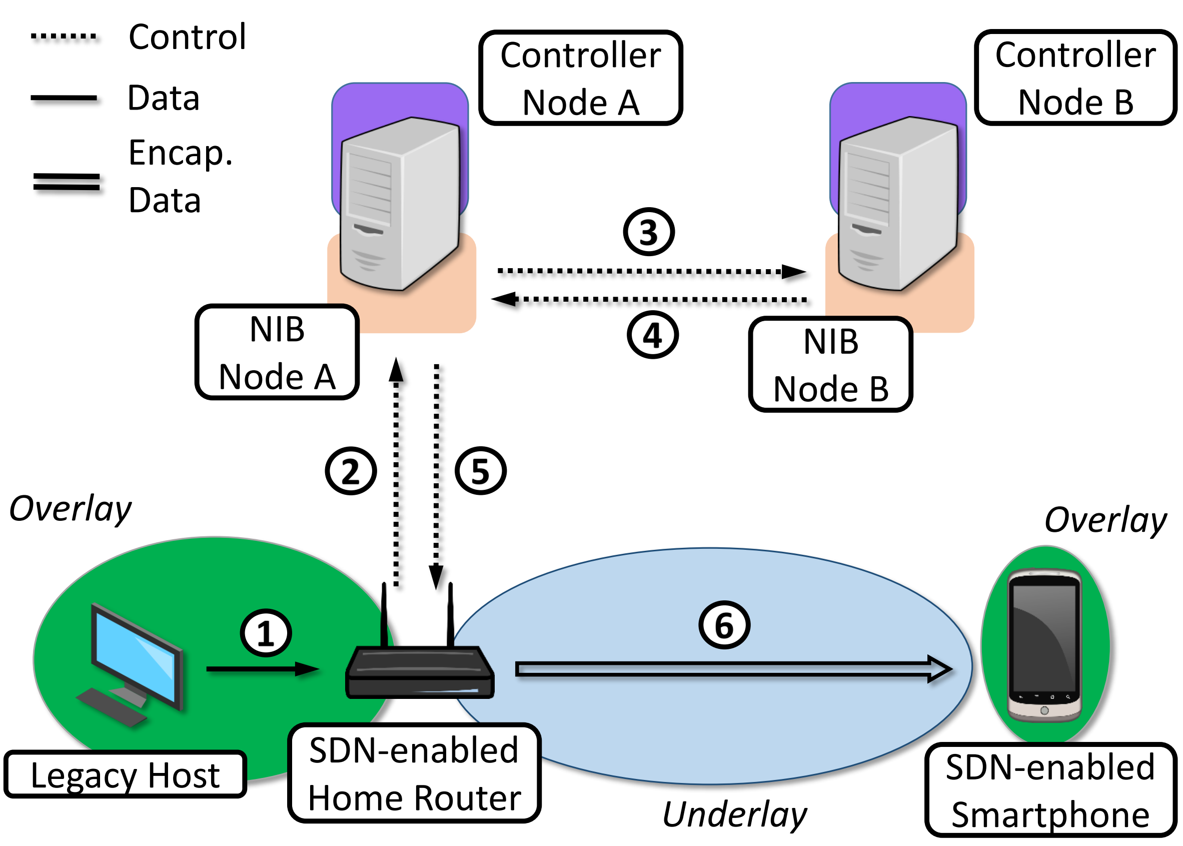


Figure 3.3 **Southbound state retrieval**

Whenever there is new overlay traffic detected at a data-plane node, the node checks its cache to get the information on how to encapsulate and forward the traffic. If there is no information available for the requested traffic, the data-plane node has to request the appropriate network state to the NIB.

To illustrate southbound state retrieval and its sequence, Fig. 3.3 shows an SDN-enabled home router encapsulating overlay traffic (1) towards an SDN-enabled smartphone. To do so, the home router requests the appropriate state to its nearest NIB node via the LISP protocol (2). In the example depicted, the required state is not stored in the queried node and thus has to be obtained from another Cassandra partition (3)(4). The state is then sent to the home router (5) which uses it to encapsulate the overlay traffic through the underlay (6).

Generally, state requests include overlay traffic source and destination. Some use-cases may include also port or protocol information, or even end-node details, such as current geo-location or wireless channel(s) in use. For most of the use-cases, the NIB tries first to find if there is a path available for the source-destination group-pair, if none is available it will then return the default ingress policies of the destination. Different use-cases may require extra information and NIB processing, but in all cases the reply from the NIB (that the data-plane node caches) only contains source-destination based state (i.e. IP granularity). Moreover, if the data-plane node is a standalone end-node (e.g. a smartphone), it can disregard source information and index just by destination.

Finally, the NIB stores the network locator of the requester to notify updates if needed. This requester data will eventually expire, therefore the data-plane node has to request periodically the state to refresh this cache or to update its stored network locator (for instance, in the case of a handover event).

While the state is being retrieved, the data-plane node can choose to either buffer the traffic (if the node has available resources) or rather encapsulate the traffic towards a re-encapsulation node. These re-encapsulation nodes have larger hardware capabilities and can offer more buffering. In addition, they are provisioned per-group and therefore they can aggregate traffic, resulting in a high cache hit ratio.

3.5.5 Data-Plane

The architecture considers all data-plane nodes alike, regardless if they are smartphones, home routers or, for instance, datacenter gateways.

End-Nodes Bootstrap

Data-plane nodes should be statically provisioned beforehand with their overlay address -or prefix- and the address of at least one controller node. How this provision is done is out to the scope of the architecture, it could be factory settings, based on node specifications or configured by the user. Data-plane node use the controller address to retrieve bootstrap configuration via NETCONF. Once the bootstrap is done, any future control communication will be done connectionless via LISP signaling. During the bootstrap, the controller provisions the nodes with an up-to-date list of NIB nodes, the list of their proxy nodes (see Sections 3.5.4 and 3.5.5), and the default egress policies for their data-plane traffic.

Overlay Encapsulation

The architecture is agnostic to the specific encapsulation format used -e.g. VxLAN (Virtual Extensible Local Area Network) [83]- and the choice must be made per use-case basis. The encapsulation chosen partially defines the rendered state that the NIB stores and the configuration to be provided during the bootstrap.

Traffic encapsulation entitles to circumvent the traffic policies of the underlay and apply overlay policies instead. Different priorities and weights can be set for underlay locators and thus overlay routing can be enforced. Moreover, SDN-controlled data-plane elements can be deployed in transit networks to help to bypass underlay out-of-scope policies. Re-encapsulating data-plane packets at these nodes enables to effectively detour traffic. In order to use them, re-encapsulating nodes are added as intermediate hops in certain path policies.

On the other hand, if a data-plane node needs to communicate to a host outside the overlay namespace it can redirect the traffic to a proxy node connected to the legacy Internet. The proxy node receives the traffic and takes care of decapsulating it and applying any NAT (Network Address Translation) operation required (if the overlay traffic is not routable in the public Internet).

3.6 Discussion

3.6.1 Use-Cases Complexity

Complex use-cases results in more database queries and potentially less parallelization of such queries. The consequence is increased latency per request and more database nodes needed. For instance, a use-case where groups depend on remaining battery and current geo-location makes necessary to obtain first the proper group prior to query for group-based path policies. The architecture has to keep a balance between the complexity of the queries (defined by the use case), the number of southbound requests (defined by the traffic and its locality), the maximum latency accepted and the number of database nodes. This trade-off has to be carefully analyzed per use-case.

In the case of the PoC presented in this work in section 3.7 the database consumes the time of one *select* operation per southbound state request. The main reason is that we use IP addresses both to identify groups and nodes. This allows us not only to use directly the information carried in the request to query the database without any pre-processing, but also to take advantage of parallelization to obtain at the same time group-based source-destination policies and destination-only default ingress policies. Other use-cases that may not take advantage of such optimization (e.g., groups depending on the remaining battery) may need two or more sequential *select* operations (first to find the group and then the group-based path).

3.6.2 NIB Scalability

From our analysis and proof-of-concept we conclude that the main bottleneck of the system is at the NIB, specifically at the database state-retrieval, given that the NIB has to support the scalability requirements of the scenario. The NIB can be overloaded by the huge amount of nodes pulling state, although this strongly depends on the characteristics of the use-case and the complexity of the associated queries. Complex use-cases will require more (and potentially more complex) queries and thus will impose more burden in the NIB.

To allocate the requirements of different use-cases, we have chosen Cassandra as the NIB back-end since it is a general-purpose database that ensures constant latency times regardless of the number of entries stored. Furthermore, it offers an operations per second ratio linear to the number of database nodes. The authors of [119] show that under a stressful benchmark with mostly read/scan operations, Cassandra is capable of keeping a latency of 20 ms and a throughput of 20K operations per second on a 4-node cluster storing a dataset with millions of entries. In a 12-node cluster the throughput goes up to almost 60K operations per second.

3.6.3 Security and Privacy

In the scenario and architecture discussed, the control of the end-nodes is partially relinquished to the operator of the SDN infrastructure (e.g the network operator, the company owning the end-device or a third party providing SDN-based services). There is a tradeoff between the benefits brought by the SDN architecture and the lack of control at the end-node. This may result in security and privacy concerns for end-users.

In general, SDN advocates for decoupling and centralizing the control. Thus, the lack of control at the data-plane devices is not exclusive to the end-node scenario. Users have to decide if they are willing to trust their SDN providers in the same way that they trust, for instance, the manufacturer of their smartphone. However, under certain conditions privacy may enhanced by adapting some of the mechanisms proposed in [124] and explained in Chapter 5.

3.6.4 Architecture Resilience

The keep-alive southbound signaling will detect failures at data-plane nodes and update the NIB state. Moreover, LISP supports active data-plane probing and if a problem is detected, nodes can fast fail-over to backup data-plane policies. NIB-level failures are covered by Cassandra background data-recovering mechanisms and ZooKeeper coordination will detect and re-instantiate any down controller or NIB node if needed.

3.6.5 Signaling Overhead

Mobility generates frequent state updates, however this has to be notified to just a few nodes: the requesters of the particular mobile node. On the other hand northbound policy changes (e.g., a new ingress policy for a datacenter) will generate a large amount of notifications, however it is expected that such changes occur less frequently. In addition, the use-cases considered in section 3.2 can operate well with partially outdated state for a certain period, this is not typically possible in existing SDN deployments.

3.7 Proof of Concept

3.7.1 Setup

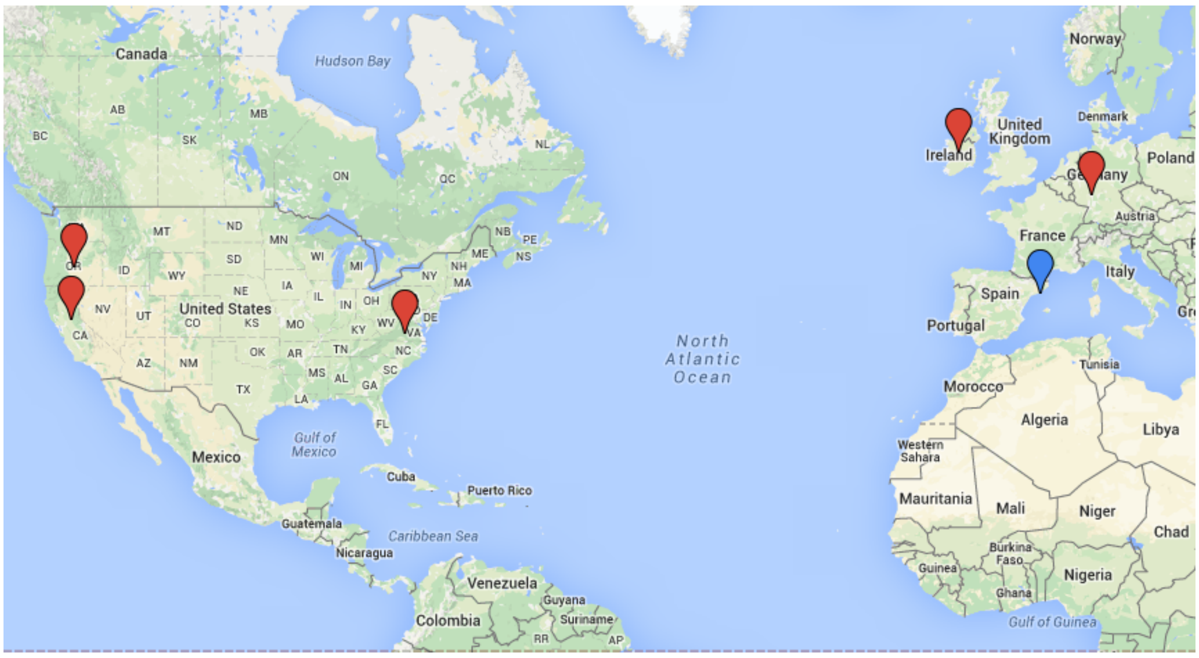


Figure 3.4 **PoC topology**

To evaluate the feasibility of the architecture, we have built a proof-of-concept (PoC) by means of allocating five virtual machines (Ubuntu 14.04, dual-core, 4GB of RAM) on the Amazon Web Services platform [1]. The machines are geographically located at Europe (Ireland and Frankfurt), US East Coast (Virginia) and US West Coast (California and Oregon). Each virtual machine hosts an instance of an in-house controller and a node of a Cassandra cluster (ver. 3.1) that stores the NIB. Additionally, to serve as end-node we deploy a laptop at our facilities at Barcelona running OpenOverlayRouter [100] (an open-source overlay software with LISP support). Fig. 3.4 shows the PoC topology, controller/NIB nodes are in red (Ireland, Frankfurt, Virginia, California, Oregon) and the end-node in blue (Barcelona). For reference, we provide the average Round Trip Time (RTT) between all the PoC locations in Table 3.2. To obtain the average RTT we measured 1K ping iterations per pair (standard deviation was less than 1 millisecond in all cases).

	Barcelona	Ireland	Frankfurt	Virginia	California	Oregon
Barcelona	-	53	47	108	181	177
Ireland	53	-	20	81	155	138
Frankfurt	47	20	-	89	166	165
Virginia	108	81	89	-	81	80
California	181	155	166	81	-	20
Oregon	177	138	165	80	20	-

Table 3.2 **Average RTT between PoC locations (in milliseconds).**

We artificially generate state and populate Cassandra for three different NIB sizes: 100K, 200K and 400K end-nodes. In all cases, the set of end-nodes is composed of -approximately- 0.2% datacenter gateways, 49.9% home routers and 49.9% mobile nodes. We use IPv4 prefixes as overlay identifiers for both abstract groups and specific nodes, in particular we select /12 prefixes for groups, and then /16 for datacenter gateways, /27 for home routers and /32 for mobile nodes. Respectively to the NIB size, we distribute the end-nodes across 100, 200 and 400 different groups.

3.7.2 Experiments

To asses the PoC implementation, we run three different experiments. First, we generate traffic in our end-node addressed to end-nodes stored in the NIB and measure the time the end-node takes to retrieve the appropriate routing policies from the NIB. We configure our end-node in Barcelona to use the NIB node in Ireland. For each of the three different NIB sizes we generate 15K unique flows. We plot the latency results as Cumulative Distribution Functions (CDF) in Fig. 3.5. The results show that the latency is independent of the NIB size, which is coherent with the discussion in Section 3.6.2. The figure also shows that the latency has a constant component (NIB processing and NIB to end-node transmission) and then a variable component, depending on from which Cassandra partition the state has to be retrieved. Latency values tend to cluster around roughly three points, that can be associated with the different delays involved to retrieve state stored in either Europe, US West or US East.

For a second experiment, we update the NIB state through the northbound and measure the time since the controller node (at Ireland) receives the northbound message until the NIB node collocated within the same VM sends the state-updated notification to the southbound. For each NIB size, we update state for 15K different end-nodes and plot the CDF in Fig. 3.6. The figure resembles Fig. 3.5 since most of the latency is due again to the network transmission delay between the different Cassandra nodes. Northbound update time is measured directly at the controller node and thus no latency is induced by the NIB to end-node transmission. This makes Fig. 3.6 to be shifted to the left around 50ms (Barcelona-Ireland RTT) when compared to Fig. 3.5. Besides, both Fig. 3.5 and Fig. 3.6 show minor latency variation across NIB sizes. This is due to the variation of network conditions over different iterations (since the PoC runs over the Internet) and to the randomness introduced during the process of generating traffic/updates.

Finally, we perform a test where we bootstrap the end-node and measure the time since it notifies its presence to the controller until the end-node acknowledges correct configuration installed. In this case the end-node uses a random controller node on each iteration. We perform 7K iterations for the worst-case of 400K end-nodes. In the bootstrap case, the latency depends not only on Cassandra delay, but also on the NETCONF communication between the controller and the end-node. This explains why the slope is softer in Fig. 3.7. Nevertheless, it is still possible to identify clustering regions, depending on how close the NIB node reached is to the Cassandra partition storing the state.

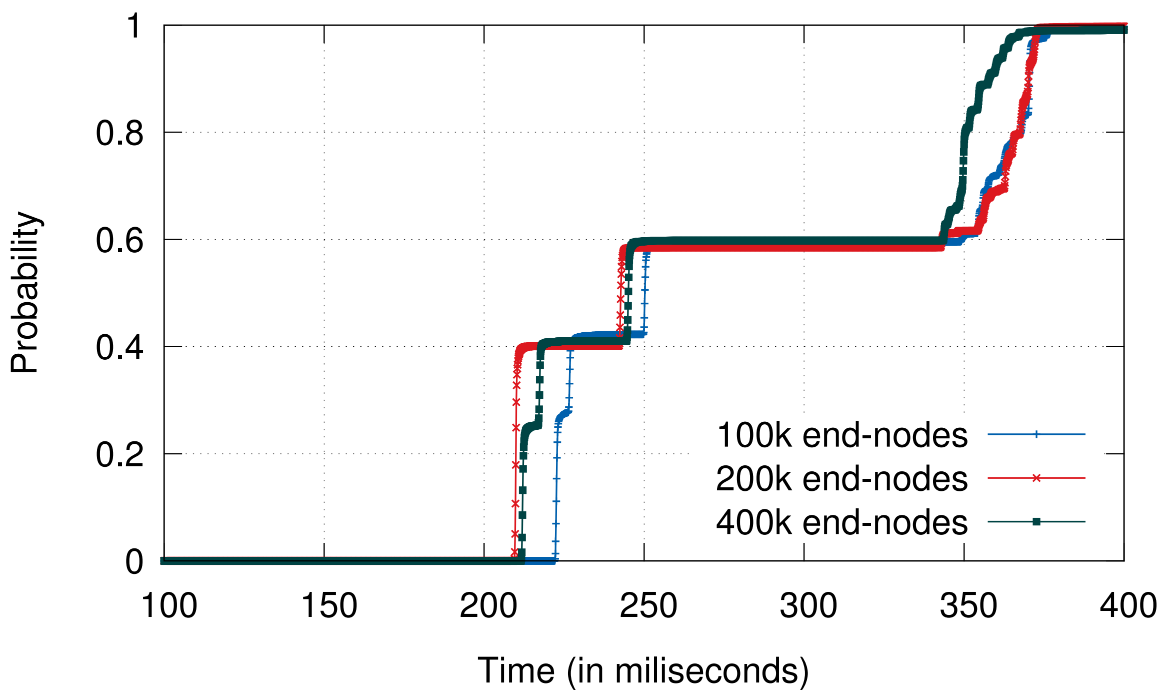


Figure 3.5 **Latency for southbound state retrieval (15K iterations)**

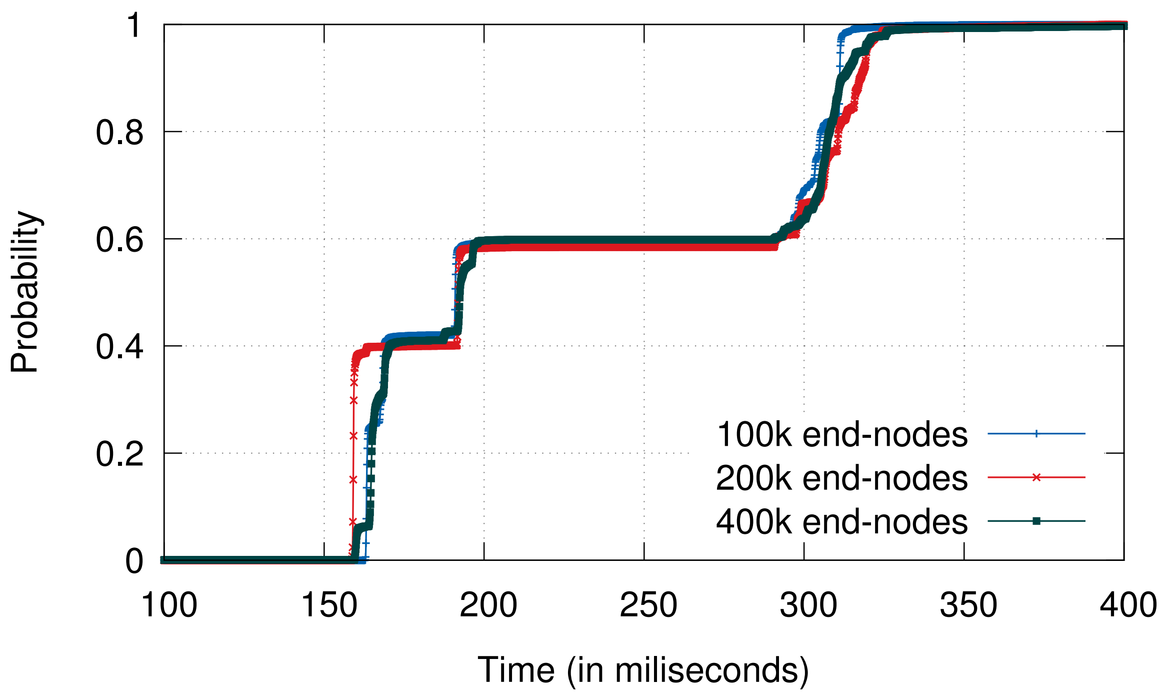


Figure 3.6 **Latency for northbound state update (15K iterations)**

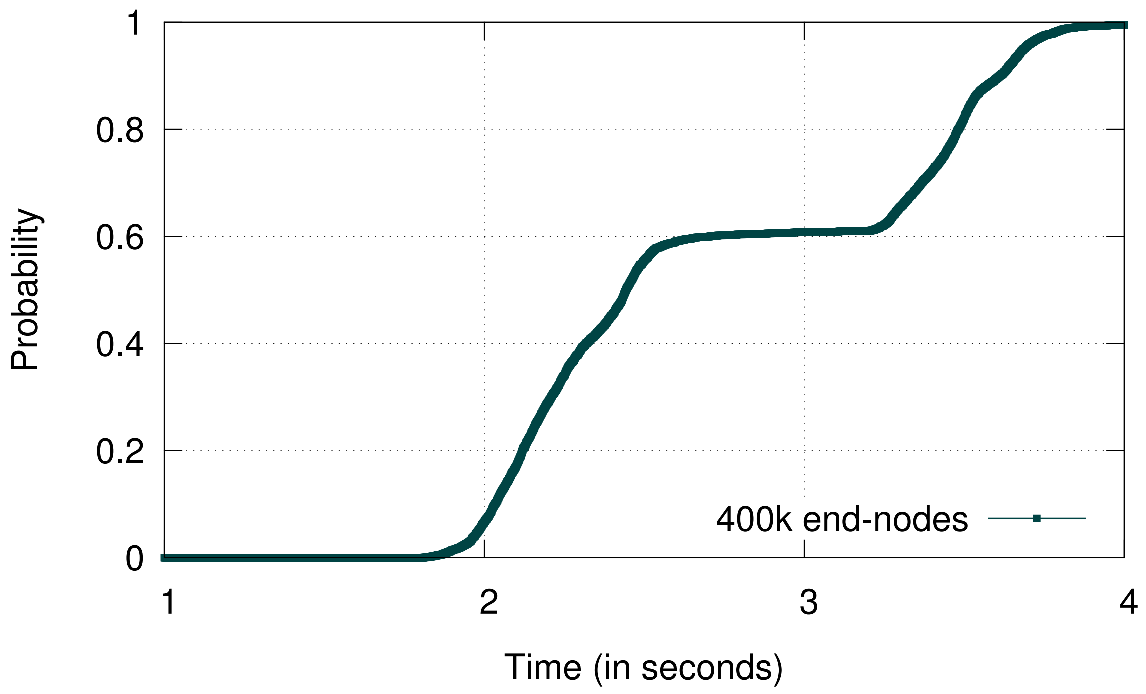


Figure 3.7 **Latency for end-node bootstrap (7K iterations)**

3.8 Related Work

There have been other works that discussed the interactions of end-nodes and SDN, nevertheless and to the best of our knowledge, this thesis is the first ever to carefully explore all implications of a full deployment of end-nodes. For instance, the authors of [143] enable SDN within a mobile device to aggregate all its available interfaces. However, and contrary to our work, they do not consider that end-nodes can be transient, scattered, with low traffic locality and very numerous. Similarly, the authors of [90] extend OpenFlow [87] to bring SDN to end-nodes and complement existing approaches in the field of Software-defined Radio. They -intentionally- kept out the scope the complete architecture to support SDN-aware nodes, which we address in this thesis.

On the field of service-centric networking, previous work has shown the value of enhancing the network capabilities of end-nodes. The authors of Serval [98] propose a service-oriented end-host stack to seamlessly change network addresses, migrate flows across interfaces or establish additional flows towards services. Despite its inherent benefits, Serval is not designed to be an SDN solution. Therefore, and as opposed to our proposal, it does not offer some SDN features, such centralized control or network programability. In a posterior work, the authors of OpenADN [108] envision a similar service-centric architecture but enhanced with an OpenFlow SDN controller. Similarly to our work, they also use an overlay approach to bypass the limitations of the underlay network. OpenADN framework focuses on switches at access networks and does not include devices at the very end of the network (e.g. smartphones). Consequently, OpenADN does not discuss OpenFlow drawbacks when the controlees are transient and highly mobile, or the challenges associated to programming highly heterogeneous data-plane devices. In our proposal we address such issues by a connectionless pull-based southbound and an intent-driven northbound.

In terms of the specific engineering solution proposed, our proposal follows some of the architectural principles of ONOS [5], a highly-scalable architecture for regular SDN deployments. ONOS proposal optimizes the system and reduces generality in order to achieve a high performance (e.g. it stores the NIB in a graph database with an optimized back-end rather than in Cassandra). Contrary, we prioritize generality to allow different use-cases and support highly heterogeneous end-devices. In the end-node scenario it is possible to keep high generality without comprising system performance given that first, the scenario does not require a real-time detailed view of the network (thus state transactions impose less burden) and second, end-nodes do not need to react as fast as typical SDN equipment to state changes (therefore more complex state processing is possible).

3.9 Conclusions

This chapter describes the specific challenges of the end-nodes scenario and why existing SDN approaches can not be directly applied to it. In that sense, we have provided a set of design guidelines for the scenario and engineered a possible architecture following these guidelines. The proof of concept shows that the architecture is feasible as long as the network operator is able to balance the complexity of the specific use-case with the requirements in terms of latency and number of NIB and controller nodes. To minimize this latency, the state has to be -partially-rendered in advance in order to decouple state rendering from state retrieval. Furthermore, this work also highlights the benefits of directly exposing the NIB to data-plane devices to let them retrieve the state they need on demand. We believe that SDN for end-nodes is a new scenario to explore and future work can be carried out to optimize the architecture presented in this chapter.

Chapter 4

Decentralized SDN Architecture for Operators NFV

The Network Function Virtualization (NFV) paradigm is rapidly gaining interest among Internet Service Providers (ISPs). However, the transition to this paradigm on ISP networks comes with a unique set of challenges, namely legacy equipment already in-place, heterogeneous traffic from multiple clients, and very large scalability requirements. In this chapter we thoroughly analyze such challenges and discuss NFV design guidelines that address them efficiently. Particularly, we show that a decentralization of the NFV control while maintaining global state improves scalability, offers better per-flow decisions and simplifies the implementation of Virtual Network Functions (VNFs). Building on top of such principles, we propose a partially decentralized NFV architecture enabled via an enhanced decoupled-state SDN infrastructure. We also perform a qualitative analysis of the architecture to identify advantages and challenges. Finally, we determine the bottleneck component, based on the qualitative analysis, which we implement and benchmark in order to assess the feasibility of the architecture.

4.1 Introduction

The Network Function Virtualization (NFV) paradigm enables software-hardware decoupling, flexible deployment of network functions and dynamic service provisioning [49]. Consequentially, it is raising special interest for Internet Service Providers (ISPs) since it will help reduce operational costs and network complexity.

However, ISP networks are challenging due to its large size, the legacy networking hardware already in-place and the heterogeneous traffic generated by ISP's customers. Therefore, an NFV architecture for ISPs must offer a platform able to scale to a wide range of different workloads and network conditions, while remaining agnostic to the Virtual Network Function (VNF) types required to process the different kinds of traffic.

These requirements dramatically increase the complexity of centralized control. Therefore, we suggest that typical NFV approaches with centralized control fall short for the ISP scale. Those solutions tend to leverage on Software-Defined Networking (SDN) [74] approaches that -logically- centralize both the state and the control. We advocate that, while the network state should be centralized, the control decisions must be decentralized and made locally.

In this chapter, we analyze these requirements and propose a set of design guidelines to address them. Based on these principles, we describe a novel decentralized architecture that offloads part of the control to an enhanced SDN infrastructure and that federates local state through a global database. This makes possible to make efficient local decisions based on global knowledge, allowing to elastically accommodate VNFs and client flows. The enhanced SDN is enabled by collocating NFV modules within the SDN controllers and then pushing the controllers close to the data-plane devices they control.

Along with the architecture we present a qualitative analysis that highlights the architectural advantages -easy VNF provisioning, enhanced flow granularity and improved scalability- and challenges -induced latency, lack of control on the underlay network and implementation complexity of the decentralized components-. Based on this analysis we identify the performance bottleneck of the system. To assess the feasibility of the architecture, we provide and evaluate an implementation for this element that we support with experimental performance results.

4.2 Scenario Requirements

ISPs deploy in-network functions to efficiently manage the traffic and offer value-added services to their costumers. The NFV paradigm allows providers to reduce both capital and operational expenses by enabling easier and cheaper deployment and simplified management of network functions. However, bringing NFV to ISP networks present a unique set of challenges. In addition to performance, manageability, reliability, stability, and security [49, 51], an NFV solution for ISP networks has the following requirements:

Compatibility with legacy hardware Usually, ISP networks are long-run deployments where there has been a significant investment in network equipment. Contrary to enterprise or public IaaS cloud datacenters which in many cases are greenfield deployments, the networking hardware of ISPs is already in place and in most cases it is not simple to update, upgrade or replace. The architecture should remain agnostic to the underlaying devices and thus be able to be deployed on top of current networks.

Support for heterogeneous traffic As opposed to the traffic observed in a data center network, the expected traffic in an ISP network will likely come from a wide range of costumers and presents diverse characteristics. Such traffic will require different kinds of processing by large sets of heterogeneous network functions.

High scalability Although scalability is a general requirement for NFV, in the case of ISP networks the size of the deployment becomes the major concern for any NFV architecture. These networks are typically very large and comprise millions of users. To accommodate such huge deployments the architecture needs to scale-out smoothly.

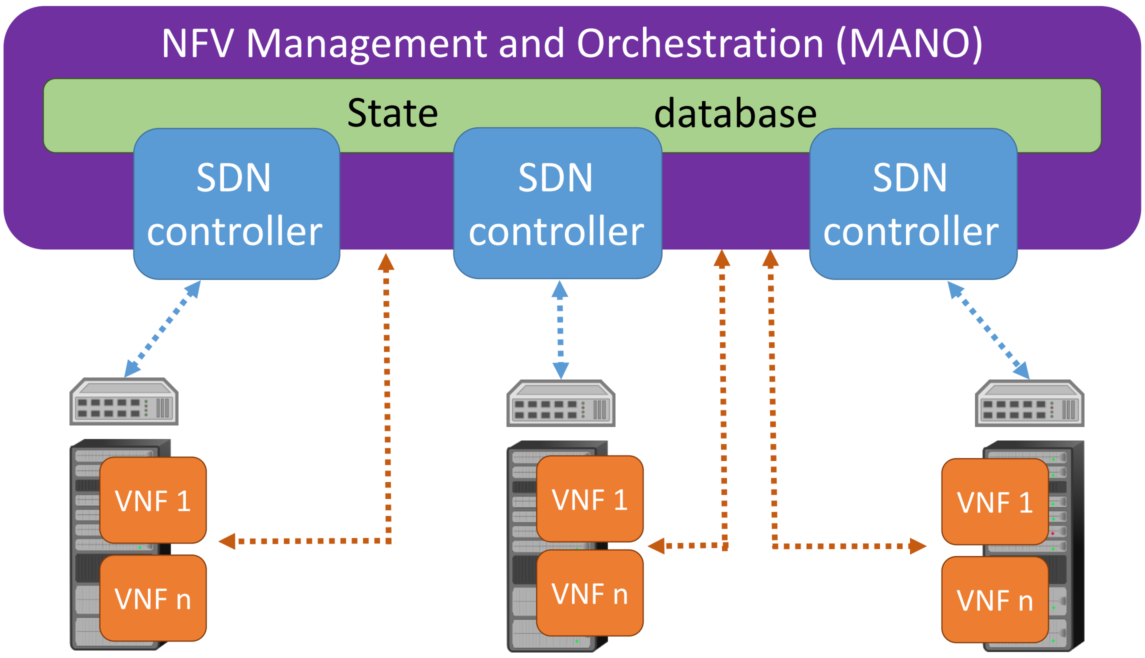


Figure 4.1 **Common centralized NFV approach**

4.3 Global State, Local Decisions

With the advent of SDN and the possibilities of control-data decoupling, network architectures usually centralize the control to ease network deployments. However, we believe that the major challenge of an NFV deployment for ISP networks is that their scale, in terms both of traffic and number of independent subscribers, increases the complexity of keeping a scalable -logically-centralized control.

In that sense, we take base on the existing work on SDN [5, 59] that proposes a distributed control, and take that approach further advocating for an NFV architecture where the control is not only distributed but also partially decentralized. We seek to find a middle ground, between the decentralization of legacy networks and the centralization brought by SDN, where the benefits of both approaches can be combined and maximized.

We argue that this optimal balance can be achieved by enforcing local decisions while keeping global state. That is, federating the state generated locally to make the outcome of the local decisions globally available. As a result, the decision on how to act on a flow can be made by the node that detects the flow, but taking into account the global state of the system at that time. This approach is possible thanks to applying the principle of decoupling state from control proposed in this thesis.

4.4 Design Principles

In what follows we propose a set of design guidelines to achieve this partial decentralization as well as to face the other requirements of the ISP scenario.

MANO disassembling We refer to the ETSI architectural framework [27] that defines the Management and Orchestration (MANO) system as the central point for NFV control [28]. The MANO system comprises the global orchestration of the architecture and the management of the VNFs and the virtualized infrastructure. Albeit this system may be -and in most cases is- physically distributed, it remains as a logically centralized point of control as shown in Fig. 4.1. We propose to keep centralized the service/functions catalogs and the general NFV orchestration, and partially the management of the virtual resources other than network (storage, computing), since this management is local per each datacenter but centralized within its locality. However, we advocate that the management of the virtual network can be completely offloaded to the infrastructure and totally decentralized and auto-coordinated thanks to an enhanced SDN infrastructure. In a similar fashion, the VNF management can be also partially offloaded. While the creation and destruction of VNF instances should be still coordinated by a centralized entity (e.g. OpenStack [106]), the monitoring, balancing and load assignment can be decentralized and coordinated directly by the enhanced SDN infrastructure. In this chapter we use the term MANO loosely and focus on the networking aspects of the system, i.e. the enhanced SDN-fabric. For the rest of the MANO components not discussed in detail we leverage on the existing solutions.

SDN infrastructure enhancement To achieve an enhanced SDN infrastructure the different instances of the SDN controller (located at the MANO system and used to control the network) must be isolated and pushed close to the data-plane devices they control, effectively enabling a decoupled-state SDN architecture. Furthermore, part of the MANO system itself should be partially distributed over those controller instances to achieve better local control, as shown in Fig. 4.2. We seek not only to put some NFV and SDN elements collocated with the data-plane nodes (as the authors of [86] propose), but also to effectively offload the control to those elements. The state database remains as part of the centralized MANO, but it is mostly updated by the decentralized controllers. Since these controllers are collocated with the data-plane devices, they have rich information about the traffic and thus can make faster and better decisions than a centralized MANO.

Offload redundant VNF functionality This decentralized and enhanced SDN infrastructure with distributed MANO modules offers a general framework where different VNF types can be allocated. This framework allows offloading redundant features of VNFs (e.g. VNF resilience, load balancing, etc) to their local MANO modules that coordinate among themselves thanks to the federated global state (see Fig. 4.2). This results in modular, optimized and compact VNFs, which enables a VNF-agnostic architecture that can efficiently handle the different kinds of traffic expected on ISP networks.

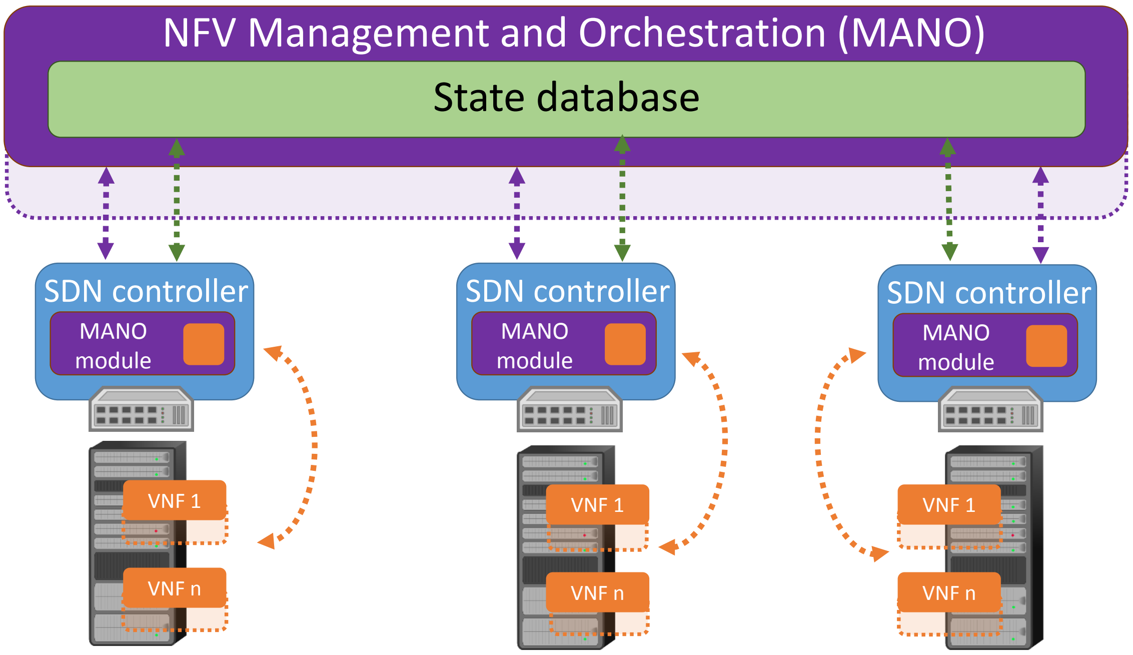


Figure 4.2 **Decentralized NFV based on an enhanced SDN**

Overlay encapsulation To overcome the constraint of the network equipment already in place, the architecture should leverage on deploying an overlay that encapsulates traffic over the legacy infrastructure. In this context, we differentiate three parts in the network, *overlay, underlay and outerlay*. The first is the virtual network instantiated by the architecture through encapsulation and the second is the legacy network beneath based on off-the-shelf hardware. We use the term outerlay to refer to the external networks that generate/receive the traffic and that connect to the overlay through enhanced SDN edge nodes. The specific encapsulation (e.g. VxLAN [83], NVGRE [40], MPLS [127], etc) used to instantiate the overlay is not architecturally relevant and will depend on the specific underlay beneath.

Strong identity-location decoupling Finally, to support the model of an VNF-agnostic overlay-based system with decentralized control, the architecture must enforce a strong decoupling of identity and location semantics and introduce different levels of indirection. In general, we aim to solve NFV challenges by moving the network appliances to the datacenter and then getting the outerlay traffic there. Identity-location split is required to maintain real time mappings of VNFs to datacenter servers, overlay traffic to underlay tunnels and outerlay clients to offered services.

4.5 Architecture

Building on the design principles discussed in Section 4.4, we propose an NFV architecture (see Fig. 4.3) that leverages on two main components. First, a set of edge nodes comprising hardware and software SDN switches and an SDN controller (provisioned with some MANO modules), and second, a distributed database that federates the state across the system.

4.5.1 Edge Nodes

Due to the legacy equipment already in place it is not cost-effective to upgrade all nodes in the network to support the required NFV capabilities. Therefore, the architecture relies on just upgrading the nodes at the network edges, i.e. the ingress/egress points of clients' networks and the ToR (top of rack) switches of the VNFs virtualization racks. Given the characteristics of ISP networks, these edge nodes should offer flow granularity for packet processing while keeping line-rate throughput on the data-plane and low-latency times for the control-plane.

To achieve this, we propose the edge node design depicted in Fig. 4.4 where an SDN controller is collocated with a hardware SDN switch to minimize switch-controller latencies. On the other hand, this hardware SDN switch is able to process the traffic at flow granularity and line-rate speed via minimizing the lookup time i.e. only performing exact match lookup over a minimal set of packet fields (e.g. 3-tuple). Any packet that does not hit an exact match entry (i.e. no rule available for its flow) is sent upwards to the control-plane through a software SDN switch. Although slower, the software SDN switch allows performing as granular flow look-ups as required (e.g. coarse 5-tuple) and define as many rules as needed. The software SDN switch contains detailed rules to classify the flow according to a given profile and hand it to the appropriate MANO service module located within the SDN controller.

These MANO modules are per-service specific software pieces that can assign flows to VNF chains and program accordingly the hardware switch (see Section 4.5.4). Once the software switch has classified the flow and handed it to the proper MANO module, the module will check if there is already cached an VNF chain suitable for the flow. If that is not the case, it uses the controller's database interface to retrieve a suitable chain from the federated information (see Section 4.5.2). If no suitable chain exists for that specific flow, the service module has to compute one itself as described in Section 4.5.4. After retrieving/computing the VNF chain, the MANO service module uses the controller's southbound interface to program (e.g. via OpenFlow [87]) the exact match rules in the hardware switch. Subsequent packets of the flow will hit the exact match entry and be processed at hardware level.

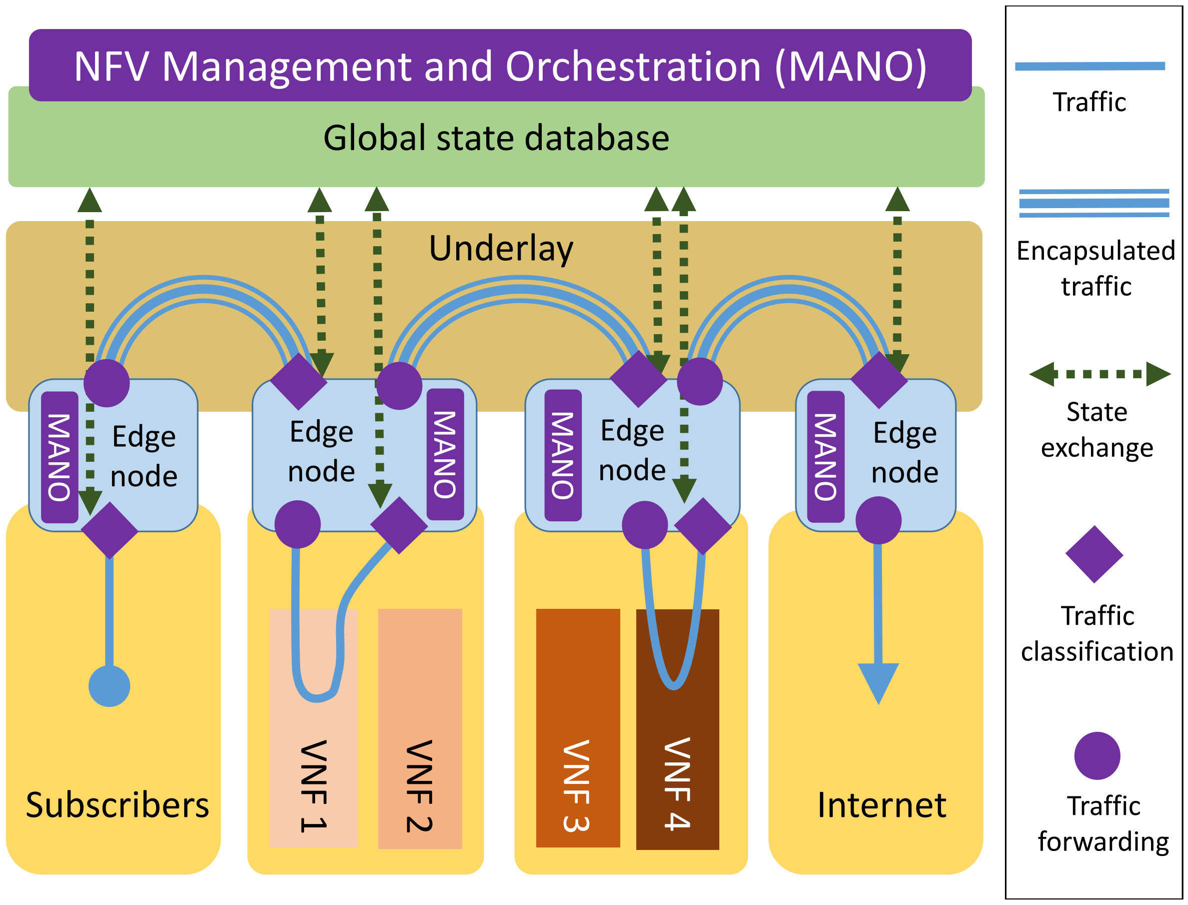


Figure 4.3 **Proposed architecture**

4.5.2 Federated Global State

A physically distributed, but logically centralized, database federates all the state generated locally at the edge nodes (e.g. computed VNF chains, etc) and makes it globally available to the whole infrastructure. It also stores general MANO information (network services catalog, VNF catalog, VNF instances, infrastructure status, etc) [28] and in this sense is populated both by the distributed MANO elements and by the centralized MANO subsystem. A summary of the information stored is provided below.

- *VNF class* $\rightarrow$ *VNF instances*: The abstract VNF classes that the different service use are instantiated into (and mapped to) specific VNF instances.
- *Flow* $\rightarrow$ *VNF chain*: Each flow already processed is mapped to its assigned chain of VNF instances.
- *VNF instance* $\rightarrow$ *Instance status*: Per each VNF instance the database stores i.e. its current location, the number of flows assigned to it, its reachability, etc.

The database follows a strong location-identity decoupling model to store the information which allows to easily introduce different levels of indirection. This entitles end-points to smoothly move across different access networks and allows VNFs to be elastically allocated both inside and outside a datacenter. Following this model, the edge nodes have to register in the global state their network location mapped to the subscribers they are serving, and the MANO system has to register the locations where it is allocating the VNFs.

For the database implementation, the architecture uses a DHT database back-end in order to offer a scalable storage with a delimited query time. In terms of available solutions, Cassandra [75] can fulfill the requirements due to its good availability and excellent scale-out capacity [119]. For the front-end interface to the database, the architecture uses LISP [30, 126], a pull-based protocol that allows retrieving identity-to-location mappings from a central repository. LISP fits well the identity-location split model required by the architecture and offers an interoperable (i.e. IETF-baked) and lightweight mechanism to retrieve state.

Given that the large size of the network leads to considerable state to store, to keep the architecture scalable the state is only pulled on demand by edge nodes. Therefore, in order to notify changes and keep the state consistent the database follows a publish-subscribe mechanism [4]. As an example, if a VNF instance has to be moved to a different physical host, the edge nodes making use of the instance will be notified in order to allow them to encapsulate the flows towards the new location of the instance.

4.5.3 Virtual Network Functions (VNFs)

The VNFs are allocated in generic virtualization racks with an edge node as ToR switch. Due to the encapsulation and the location-identity split, the VNFs can be dynamically moved across hosts, racks or datacenters. Therefore, the model allows to both encapsulate the traffic towards where the VNFs are and/or move the VNFs close to where the traffic is. All this path computation complexity is offloaded to the MANO service modules at the edge nodes.

In this architecture, the VNFs are unaware of the rest of the system (i.e. they do not know the next hop for a flow) therefore the scope of the VNF state is restricted to flow processing. The fact that the VNFs are unaware of each other simplifies their elastic allocation and the deployment of new services, since different VNFs from different services can be easily chained.

The architecture keeps VNFs as simple as possible. Ideally each VNF should do only one single task. The idea is to be able to create complex services by chaining several VNFs while maintaining a flexible system that can scale-out in a modular fashion. That is, if a VNF is experiencing high load, that specific VNF can be scaled-out independently without affecting other VNFs in the chain.

In this sense, the architecture trims out redundant logic common to all VNFs and moves it to the distributed MANO modules. Scalability, load balancing, high availability, etc, are decoupled from the VNFs and offloaded to the infrastructure. For instance, a VNF processes packets unaware of any balancing policies and it is its local MANO module that monitors the load and takes care of tuning the general system to properly balance flows among different VNFs.

The combination of an architecture that is VNF-agnostic and of VNFs that are simple, light and interoperable, enables VNF outsourcing. The VNFs do not need to be specifically developed for the particular NFV system but rather can be developed by third parties and smoothly integrated with other VNFs. The architecture eases the development of such outsourced VNFs, since VNF-vendors can leverage on the mechanisms offered by the infrastructure and thus avoid dealing with ISP networks scalability or availability requirements.

4.5.4 Management and orchestration (MANO)

The architecture leverages on a service-based model where the centralized MANO installs distributed service-specific modules in the edge node controllers, and programs the software SDN switches to redirect service-specific traffic to the proper MANO module. Services use the centralized MANO subsystem to create the VNFs, but through these distributed modules, each service defines the traffic to be processed by the service, the VNF classes that should process

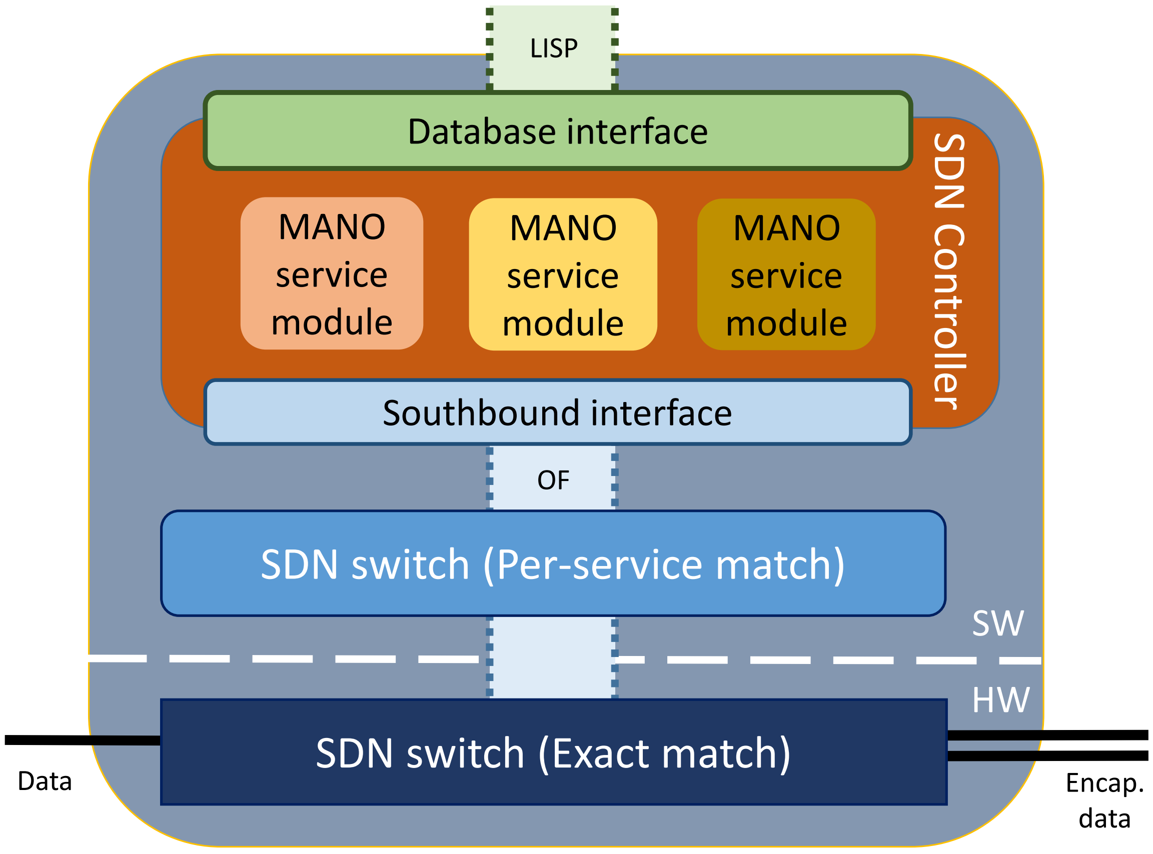


Figure 4.4 **Edge node internals**

the traffic, how to build on-demand VNF chains based on available VNF instances and how to re-allocate and move the traffic and the VNFs.

To program the hardware switches at the edge nodes, the decentralized MANO modules need to know the VNF chain assigned to a flow or compute one if none exists. The service description defines the classes of VNF to chain, but the service implementation per module decides which is the best VNF chain among all possible VNF instances. For instance, for a real-time analytics service the best chain may be composed of VNF instances placed in low-latency locations while for an on-the-fly video decoding service the chain may comprise the currently less loaded VNF instances. A computed VNF chain is stored in the database to make it available globally and cached locally to assign it to similar flows in the future. The distributed service-specific MANO modules monitor the traffic and the VNFs and are synchronized with the federated global state and with the central MANO subsystem. Therefore, they can reassign flows to different chains or recompute chains if required. Such changes will be notified to the rest of the system as described in Section 4.5.2.

4.6 Qualitative Analysis

The proposed architecture results in a set of qualitative advantages and technical challenges that we analyze in this section.

4.6.1 Advantages

Decentralization boosts scale-out

Since the coordination required among the different parts of the architecture is relaxed, it is easier for these parts to scale-out independently. This can be achieved for the architecture as a whole (e.g. adding more edge nodes) or for each component individually (e.g. adding more physical servers to an edge node cluster).

Flow granularity even at large networks

The optimized flow lookup allows for more flows to be handled per each hardware switch and thus reduces the cost of scaling-out the edge nodes to allocate more traffic. In general, all architecture components are designed to keep flow granularity despite the network size. Edge nodes process flows in parallel independently, VNFs keep only per-flow state and the federated database uses a plain namespace with constant access time.

Better per-flow decisions

The decisions on how to process a flow are taken close to the data-plane devices carrying the flow itself. Therefore, more and richer per-flow information is available. The flow granularity processing and this detailed per-flow information entitle for complex per-flow decisions, something that is challenging to accomplish with traditional logically centralized architectures.

Simpler VNFs and VNF outsourcing

Offloading redundant network functions features to the enhanced SDN infrastructure enables to reduce VNF complexity. This combined with the VNF *agnosticism* enabled by the architecture offers the possibility of outsourcing the VNF development to third parties.

4.6.2 Challenges

Global state query latency

The architecture forces edge nodes to retrieve state from the global database (e.g. to check if there is already a VNF chain associate to a flow) and to keep this state updated. Although state queries are done in parallel (i.e. they do not affect architecture's scale-out properties), the architecture needs to ensure that this state retrieval is fast enough in order to avoid being a burden for the system performance.

However, technologies already available should offer low enough query time. In particular, the *read* operation of a Cassandra (or equivalent) DHT database will add no more than a few milliseconds even under high loads [119] and an optical underlying transport connecting the edge nodes to the database will only induce latency in the order of microseconds [64].

Lack of control for the underlay network

The architecture uses the underlay network and has to rely on its correct operation. If that is not the case the architecture has no control over it and is unable to fix the problem.

However, ISP networks will likely have their own troubleshooting and healing mechanisms, as assumed in [59], and be reliable enough to allow the correct operation of the architecture. Even though, in the case of a major connectivity problem, the enhanced SDN infrastructure may be able to transparently redirect the traffic to a reachable point thanks to the identity-location split schema enforced.

Edge node implementation

The proposed service-based decentralization presents a particular challenge at the edge nodes since it requires complex classifying and forwarding mechanisms that need to remain scalable.

For the embedded controller, each MANO module is independent of the others and its performance is not affected by the number or complexity of other modules and therefore scale-out requirements can be met with a cluster-friendly controller (e.g. OpenDaylight [103]) able to distribute the load. On the other hand, the hardware switch is agnostic to the service complexity or its number since it only considers independent exact match rules, and thus it can be scaled-out across several hardware devices.

The bottleneck of the system is the software switch. In this case, contrary to the rules allocated in the hardware switch, the rules required to support more services or more complex ones comprise wildcarded fields, longest prefix match lookups and different priorities (since these rules will likely overlap). This makes the complexity of flow classification at the software switch to increase non-linearly with respect to the complexity or number of services. As a result, the architecture needs a software switch capable of achieving the linear scalability required by the large number of flows expected, despite the non-linear complexity faced in the flow classification. This makes the software switch the greatest challenge of the architecture.

4.7 Software Switch Implementation

From the analysis on Section 4.6, we conclude that the scalability on the system will be capped out by the performance achieved by the software switch. In the related research literature it is possible to find different performance evaluations for software switches reporting different results [114, 145, 23, 111]. To asses the feasibility of the architecture, in this section we discuss the software switch implementation.

We measured the performance of currently available software switches, particularly Open vSwitch [111], and we were able to achieve 11M packets per second (pps) using OVS 2.3.1 (DPDK-optimized [58]) on a single core with 100 OpenFlow rules and less than 100 traffic flows. This number is similar to the one reported in [23] and, at the best of our knowledge, this is due to caching lookup results for known flows to effectively bypassing the OpenFlow lookup tables. When we raised the number of flows to 500K, a number closer to the ISP scenario, the performance dropped to 300K pps. We observed also non-linear scaling since a 8-core configuration only achieved 1.2M pps. The hardware used for these tests was similar to the one described latter in this section.

To achieve ISP performance requirements we implemented our own in-house software switch, written in C and leveraging on DPDK. It is based in a multithreaded design where all threads have access to the rules from a common memory space. Each thread handles a subset of the flows distributed to it based on 5-tuple hashing performed by the NICs, using Receive Side Scaling (RSS) technology with a queue per thread. The OpenFlow tables are presented as static tables and updates are performed on a shadow copy of those tables. Periodically the shadow copy is switched with the active table set, updated with the changes made on the shadow copy and from there updates commence on the new shadow. The key to the high performance implementation is that for every packet, the relevant rules are fetched into cache memory just in time for lookup. By pipelining the rule prefetching (i.e. handling a few packets in parallel by each thread) the throughput is achieved by effectively always referencing rules that reside in the CPU cache (and not in off-chip memory). As a result, the performance is almost independent of the number of rules.

To measure the scalability of the proposed software switch we performed the following benchmark. We ran the switch on an Intel-based server with dual Intel Xeon E5-2690 2.9GHz

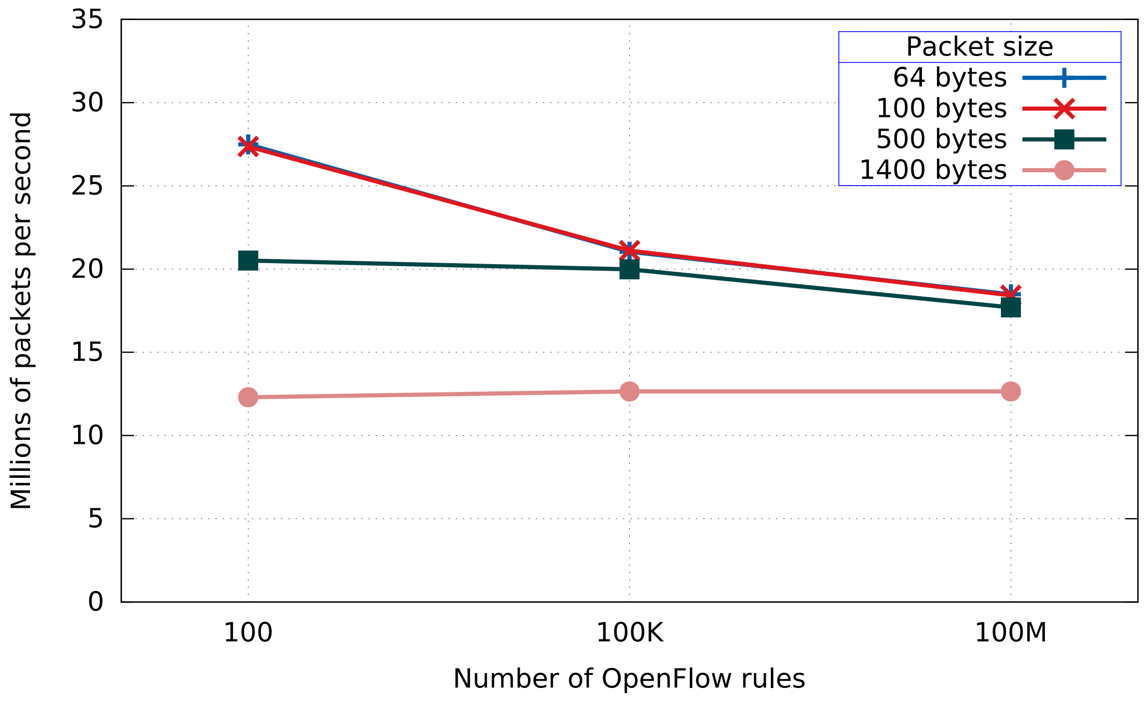


Figure 4.5 **In-house software switch performance with millions of rules**

8-core per socket CPU (i.e. 16 total cores) with 128GB of RAM and a set of 16 interfaces of 10Gbps each. We populated the switch with rules ranged from 100 to 100M and we generated traffic evenly distributed across all rules (i.e. the traffic was forged to hit all rules at the same rate). Fig. 4.5 shows the packets per second processed by the switch for different numbers of rules and packet sizes. In all cases the delay per packet was constant and around 50 μs. The figure shows how the switch scales almost linearly and achieves the requirements of the architecture.

4.8 Related Work

Recent research directions seem to point to an interest of the community into achieving better control of the underlying resources, however there is a lack of an architecture able to fulfill the requirements of all the different research proposals. In this sense, the architecture discussed in this chapter can fill this gap with its decentralized approach and its autonomous NFV modules collocated in the SDN controllers. Moreover, it can lay the foundations for the advent of this new generation of architectures where the NFV control is thigh coupled with the SDN infrastructure.

An early example on how to enhance SDN data-plane devices by pushing some control to them can be found in the work of *Risso et al.* [121]. The same authors later proposed in [14] an extended framework where they describe a dynamic NFV architecture targeting operator networks. In our work we move forward on that direction by collocating all the SDN control with the data-plane elements.

One example of such research is the work by *Gember-Jacobson et al.* [42] where they design an API to allow network functions to coordinate among themselves in an NFV-SDN scenario to offer better packet processing. The decentralized MANO modules of the presented architecture can intercommunicate with API-enabled network functions and thus enable enhanced flow processing.

Another example of this research trend is the work of *Matias et al.* [86] that propose to leverage on the SDN capabilities of the network to directly implement some VNFs without the need of allocating a VM with the VNF. For instance they argue that a firewall can be implemented by means of OpenFlow rules installed in the networking devices that drop (or allow) packet belonging to target flows. This proposal can use the architecture presented in this chapter to efficiently program the edge node switches to implement SDN-based VNFs.

Beyond academic initiatives, industry-driven projects like Telefonica's OpenMANO [105] also shows an interest in modifying existing NFV solutions to, among other goals, achieve a closer control of the network fabric. To that end, they modify OpenStack to better control a networking infrastructure based on Intel's DPDK. The architecture that we propose can similarly program a DPDK data-plane in the same way it programs the edge node hardware switches.

4.9 Conclusions

The architecture presented in this chapter aims to address the scalability concerns of large ISP networks via partially decentralizing the MANO system. Contrary to most NFV proposals, the SDN controllers used by the MANO are collocated with their controlees and provisioned with NFV modules. This enables faster local processing by means of reducing centralization. In general the architecture seeks to find a good tradeoff of complexity, performance and scalability by decentralizing some components while keeping a centralized state. This results in a set of benefits for NFV deployments on ISP networks such as, enhanced scale-out capacity, flow granularity decisions and optimized VNFs. However, these benefits come at the cost of some associated challenges, particularly the egde node complexity introduced by the decentralization. We identified the software switch needed for flow classification at the edge nodes as the bottleneck of the system and proposed a switch implementation able to fulfill the performance requirements of ISP networks.

Part II

Deploying SDN for End-Nodes

Chapter 5

Privacy for LISP Mobile Nodes

LISP offers native mobility by decoupling IP addresses semantics into Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). LISP Mobile Node (LISP-MN) is the particular case of LISP that specifies mobility. Supporting LISP-MN is a key point to enable SDN for end-nodes in the way described in Chapter 3. However, mobility protocols have an inherent issue with privacy since some users may not want to reveal their location or their identity. In this aspect, LISP-MN is no different from other mobility protocols and thus privacy issues should be considered to effective deploy an infrastructure relaying in LISP-MN. In this chapter, we review LISP-MN and its privacy concerns and propose solutions to enable privacy, both in terms of location and identity.

5.1 Introduction

In the LISP architecture, EIDs identify hosts and are assigned independently of the network topology while RLOCs identify network attachment points and are used for routing. LISP design allows EIDs to remain unchanged even if a topological change (such as a handover) occurs. This makes the protocol well suited for mobility [125, 39]. Indeed, the mobility version of the LISP protocol (LISP-MN [32, 125]) proposes LISP-enabled endpoints, which enables legacy applications to smoothly roam across access technologies and service providers. The LISP-MN protocol uses the Mapping System [38] to disseminate the EID-to-RLOC bindings.

Since mobility protocols typically use addresses to locate users, they raise privacy concerns, and in this context LISP-MN is not an exception. An attacker could learn the (approximate) physical location of a user by monitoring its locator address, for instance by using IP geographical localization techniques [142]. This issue is exacerbated in LISP-MN when compared to other mobility protocols, such as Mobile IP [109, 63]. In Mobile IP an attacker has to establish a connection with the mobile node to learn its location, this way a mobile node can reject inbound connections from untrusted peers. However, in LISP-MN an attacker has just to query the (publicly accessible) LISP Mapping System to learn the location (RLOC) of a user, which is beyond its control.

In addition to location privacy, anonymity is of an increasing concern as well for a subset of today's Internet users. As a result of these concerns, the industry is developing mechanisms to improve online anonymity. For instance, some popular web browsers include a *private browsing mode*, where tracking cookies have the lifetime of a single browsing session, and a ”Do Not Track” option to opt-out from advertising network behavioral tracking. However, a LISP-MN host still discloses its unique EID even in these browsers operating mode, making EID based tracking possible. Given the fact that an assigned EID rarely changes (e.g., a mobile phone number), it can be easily associated to the user's identity and might be desirable to not disclose it in order to protect user's anonymity.

In this chapter we discuss how LISP-MN can address both issues: location and identity privacy. It is important to note that we take a realistic approach when extending LISP-MN, since we aim to propose *deployable* solutions, and minimize the changes to the main LISP protocol. Although some parts of the LISP protocol are currently under development, some of its elements and specifications are not trivial to redefine and hence, we want to minimize the changes to the main LISP protocol specification. Further, we also analyze the level of security achieved with the proposals that appear in this chapter, their required trade-offs and the feasibility of their implementation. Finally, we evaluate the burden introduced by the proposals in both the data and control planes.

5.2 Background: LISP-MN

To put this work into context, in this section we provide an overview of the LISP-MN specification, its particularities regarding the main LISP specification and how it relates with other mobility protocols, such Mobile IP [109, 63]. LISP Mobile Node (LISP-MN) [32] is a lightweight implementation of the LISP protocol intended for mobile use. It uses LISP mobility features to build a LISP-based mobility architecture and protocol. Separating the host identity (EID) from its locator (RLOC) enables seamless endpoint mobility by allowing the applications to bind to a permanent address while the RLOC of the host can change many times during an ongoing connection.

In LISP-MN, a Mobile Node (MN) is typically statically provisioned with an EID that is used for all its connections, to enable applications to bind to a static address. The current point of attachment to the network defines the current RLOC for the MN and is subject to change over time. LISP-MN allows the location of the host to change without breaking the transport layer connection. This is possible since the MN implements a lightweight LISP tunnel router that performs mapping resolution and encapsulation operations directly on the MN. Packets - except for management protocols such as DHCP - are LISP encapsulated by the lightweight LISP tunnel at the mobile node, and routed based on the RLOCs to the destination site. The mobile node tunnel routers remove also the LISP header from incoming packets before sending them to upper layers to ultimately reaching the destination application. In the event of a handover (e.g. when the location of the MN changes), the MN receives a new RLOC from the network and it updates its EID-to-RLOC bindings at the Mapping-System to maintain reachability at its new location. Thanks to those updated bindings, other LISP tunnel routers can learn the new RLOC of the MN. More details and discussion of LISP-MN can be found in [32, 125, 91].

Fig. 5.1 summarizes the basic operation of LISP-MN. In the figure, the MN wants to communicate with its peer, from which only knows its EID. It sends (1) a Map-Request (MRq) to obtain the RLOC of the peer. This MRq is routed (2) through the Mapping System to finally reach (3) the Tunnel Router (xTR) of the LISP site where the peer is. The xTR replies (4) to the MN with its RLOC in a Map-Reply message (MRp). Finally, the MN sends (5) the data to the xTR which forwards (6) it to the peer.

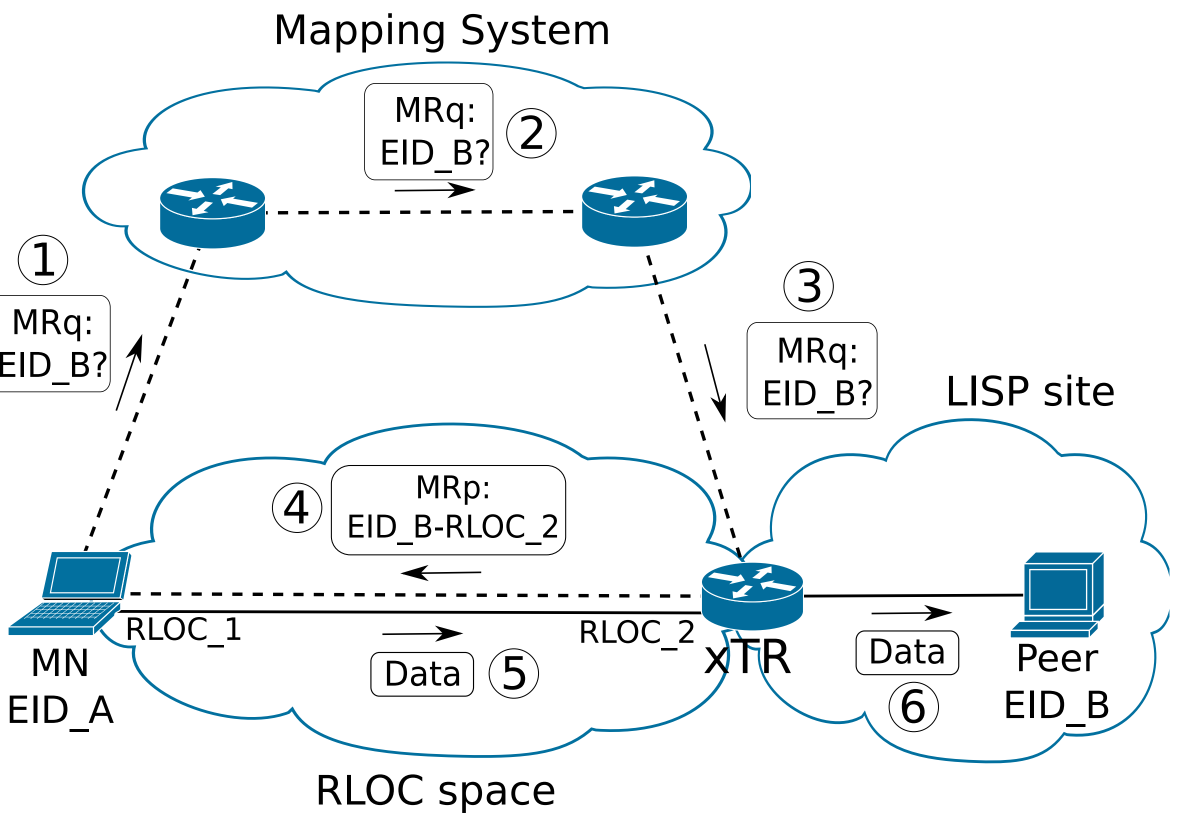


Figure 5.1 **LISP-MN Overview**

5.2.1 LISP-MN specifics

There are a few LISP specific aspects that need to be carefully considered in the LISP-MN mobility scenario. On one hand, the handover process (e.g. switch network access point) for a LISP-MN provisioned mobile node happens as follows. When a MN changes its attachment it regains connectivity in a new subnetwork. It first obtains a new RLOC and notifies the new EID-to-RLOC binding to its Map-Server. The MN has to update also all the bindings stored in the Map-Cache of the peers, either LISP routers or nodes, with which it is communicating. In order to do it uses the special signaling message called Solicit-Map-Request (SMR). The reception of such message triggers a Map-Request to refresh the binding, it is important to note that such message is transmitted over the Mapping-System, and hence prevents the double-jump problem (i.e. two MNs changing location at the same time). Overall, the handover latency in LISP is 1.5 Round Trip Times. Research efforts to optimize such handover latency are already under development and take base on LISP-SEC [84].

On the other hand, in the case of a LISP-MN mobile node behind a LISP site, double encapsulation is required. The RLOC of the MN is in fact an EID assigned by the xTR in charge of the LISP site. We refer to this EID as Location EID and to the permanently assigned MN's EID as Permanent EID. In this scenario, when the MN roams into a LISP site, the xTR assigns (by means of DHCP) the location EID. The MN registers this EID into the Mapping-System as its RLOC. The peer that wants to communicate with the MN queries the Mapping-System and receives the Location EID as the RLOC of the MN. The peer will then realize that the Location EID is not routable and will consequently query again the Mapping-System to obtain the RLOC, this is the IP address assigned to the xTR where the MN is attached. At this point the peer (typically an xTR, or another MN) will then double encapsulate the packets towards the MN. Upon reception, these packets will be first decapsulated by the xTR, and then by the MN.

5.2.2 LISP-MN vs. Mobile IP

Leveraging on LISP's locator-identity decoupling, LISP-MN is a fully featured mobility protocol that supports both IPv4 and IPv6, network mobility, route optimization and native multihoming. Compared to other identity-location split schemes, LISP has a unique position since it is incrementally deployable and it does not require changes to transport/application implementations. This makes LISP-MN a real alternative to provide mobility to the Internet.

During the last decade we have witnessed huge progress in wireless access technologies. This has lead to an increase in the adoption of mobile devices and demand for mobile Internet. Consequently, a plethora of solutions and protocols have been proposed (see [22] and the references therein). Arguably, the most popular is the Mobile IP [109, 63] family of protocols. LISP-MN has a set of features with respect to Mobile IP that makes it an interesting alternative. In the following we detail the main reasons that support such statement:

- The Mobile IP family of protocols is a set of protocols that provides basic and advanced functionalities to mobile nodes. For instance there are two separate versions for IPv4 [109] and IPv6 [63], and additional protocols for network (prefix) mobility. Furthermore, advanced features such as fast handovers [69] or multihoming [137] are again defined in separate specifications. As a result, a developer willing to adopt Mobile IP must read and implement different protocols defined in several specifications. In LISP-MN all the features are defined in a single protocol, and thus only require a single implementation. Currently LISP-MN supports IPv4, IPv6, network mobility and multihoming natively. The benefits of the native multihoming of LISP are inhered by LISP-MN: each EID prefix can be mapped to more than one RLOC, and each RLOC can be assigned specific priority and weight. This simplifies adoption and reduces significantly capital expenditure costs.

- LISP-MN separates the control plane functionality from the data plane, allowing each to scale independently. Since LISP-MN does not require Home Agent or Foreign Agent network elements in the data plane, it avoids triangle routing at the data plane level, for both IPv4 and IPv6 address families, along with network mobility data packets always follow the shortest path and hence, in this context LISP-MN incorporates native route optimization support. It is worth to note that when communicating with non-LISP sites, communications must be forwarded through a proxy.

- Separation of control plane from data plane in LISP-MN facilitates the decoupling of end-point identity from the mobility service provider. The sole functionality of the control plane is to locate a mobile node, much like DNS locating a service or a host name today. Similar to DNS, LISP control plane has a distributed and federated Mapping-System. Mobility becomes a native feature of the network architecture and avoids mobility provider lock-in. In this context, LISP-MNs can change their RLOC provider -typically an Internet Service Provider (ISP)- and are tied to their EID provider, however and unlike in Mobile IP, such EID provider may not be an ISP (i.e, the home of the mobile node) but rather a third-party company. EID providers do not operate in the data plane as Home Address providers in Mobile IP, but in the control plane, and represent new business opportunities.

5.3 Privacy in LISP-MN

After describing the LISP-MN protocol in the previous section, in this section we discuss the privacy concerns of the protocol and propose solutions to provide both location and identity privacy. Although we present different solutions that address these issues independently, both proposals can be combined to provide full privacy to LISP-MN.

5.3.1 Location Privacy

Location privacy is a well-known problem in mobility and the most common solution is to use a trusted proxy. This way the proxy forwards the traffic from the MN and only the locator of the proxy is exposed. The LISP architecture offers proxies called RTRs (Re-Encapsulation Tunnel Routers) that can be used for this purpose. The RTRs receive LISP traffic, decapsulate it and rather than forward the traffic to end-hosts, they lookup in the Mapping System for an appropriate next LISP hop and re-encapsulate the traffic towards it. They serve in LISP deployments to provide Traffic Engineering possibilities [31] and NAT Traversal capabilities [25] to LISP nodes.

In order to achieve location privacy for LISP-MN using an RTR proxy, we can leverage on the NAT traversal mechanism. NAT traversal in LISP is needed since LISP control and data messages are UDP encapsulated and they use the destination port 4342 and 4341 respectively. However, without prior configuration, NAT-boxes do not allow incoming packets addressed to these ports. The operations of a MN behind a NAT are as follows.

First the MN must check weather it is behind a NAT box. To obtain this information the MN sends a special signaling packet (Info-Request) to its Map-Server. In turn, the Map-Server replies with a list of available RTRs and the actual source address and port of the MN (i.e. the ones that the Map-Server sees in the packet it receives). With this, MNs are aware that they are (or not) behind a NAT box. In the NAT case, the MN sends a Data-Map-Register message to the RTR with source port 4341. This message is an encapsulated Map-Register which opens the port 4341 in the NAT table and it is used by the RTR to infer the translated RLOC and port of the MN. Then, the RTR forwards the Map-Register to the appropriate Map-Server. Please note that this Map-Register contains the RLOC of the RTR, not the one of the MN, so all MN's incoming packets are received by the RTR which, in turns, forwards them to the translated RLOC and port of the MN. This way the RTR acts as a signaling and data-proxy for the MN. Further details on NAT-traversal for LISP and LISP-MN can be found in [25, 67].

Figure 5.2 covers the part of the NAT traversal mechanism relevant to this chapter, i.e. the negotiation of the RTR and the traffic detour. The MN requests and gets a list of available RTRs from the Mapping System (1), the MN selects one of them and configures the RTR as its

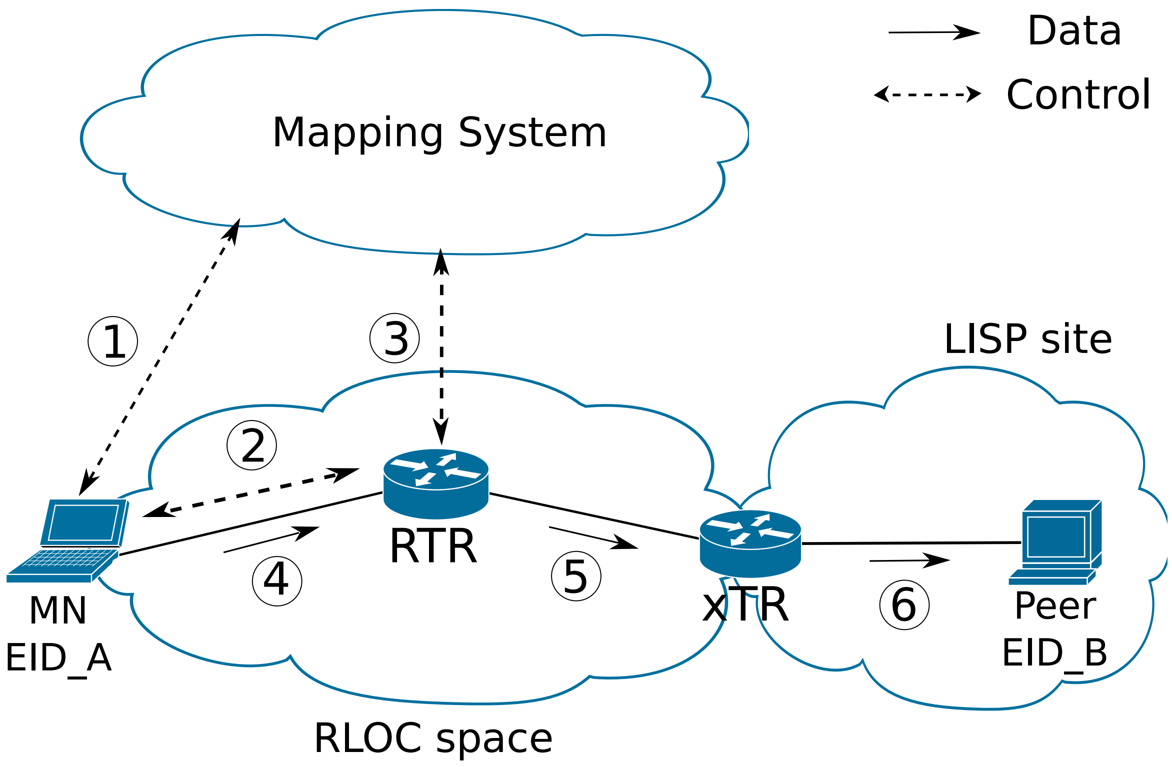


Figure 5.2 **LISP-MN using an RTR proxy**

network attachment point (2, 3). From that point on, the MN detours all its traffic towards the RTR (4) and therefore remote nodes receive the traffic from the RTR (5,6) and not from the MN. The traffic follows the same path on its way back to the MN.

Due to the presence of the RTR on the path, the location of the MN is guaranteed to be private during the NAT traversal session. An MN willing to hide its location can trigger the NAT traversal procedure, and thus force its traffic to go through an RTR, even when it knows there is no NAT present. Moreover, for the specific purpose of location privacy, the NAT traversal procedure can be improved as follows.

We propose to add a flag to the control messages exchanged on the bootstrap of the NAT traversal mechanism to notify that the procedure is going to be used to achieve location privacy (it might or might not be used also for NAT traversal). The extra flag allows the Mapping System to know that it should send, alongside the list of available RTRs, extra privacy-related information. This could include the geo-coordinates of the RTRs, their the current load, the probability of an attack on each on them, etc. The exact information included is up to the specific implementation, but it can be encoded in the LISP control messages using the format defined in [33]. The MN can use that extra information provided by the Mapping System to choose the most suitable RTR for its needs, e.g either choose a close RTR to reduce the latency or choose a far away one to better mislead possible attackers.

A company interested in offering location privacy to its costumers can deploy a set of RTRs in the Internet. In order to access the RTR the MN requires a pre-shared key, in a similar way it needs one to register to its Map Server [37]. This pre-shared key, that grants access to the RTR, can be used to enforce that the client is paying for the service. The company has incentives to deploy more RTRs, and more importantly, with a good global coverage. This will reduce the routing inefficiencies of private communications and provide more deceptive locations to offer a better service to the subscribers. With this in mind, the company that invests in more well placed RTRs will be more competitive.

5.3.2 Identity Privacy

In this section we extend LISP-MN to offer identity privacy, the main purpose being to hide the EID to untrusted peers. A classic approach on legacy IP networks to deal with identity protection is to use temporary IP addresses [96], we take base on that concept and propose to use temporary identifiers rather than the real one. This section proposes two different approaches to provide such temporary identifiers on a LISP-MN deployment, a MN-driven infrastructure-less solution and a solution dependent on the deployment of a new element. It is important to note that in both cases the identity privacy can only be offered when the MN initiates the connection.

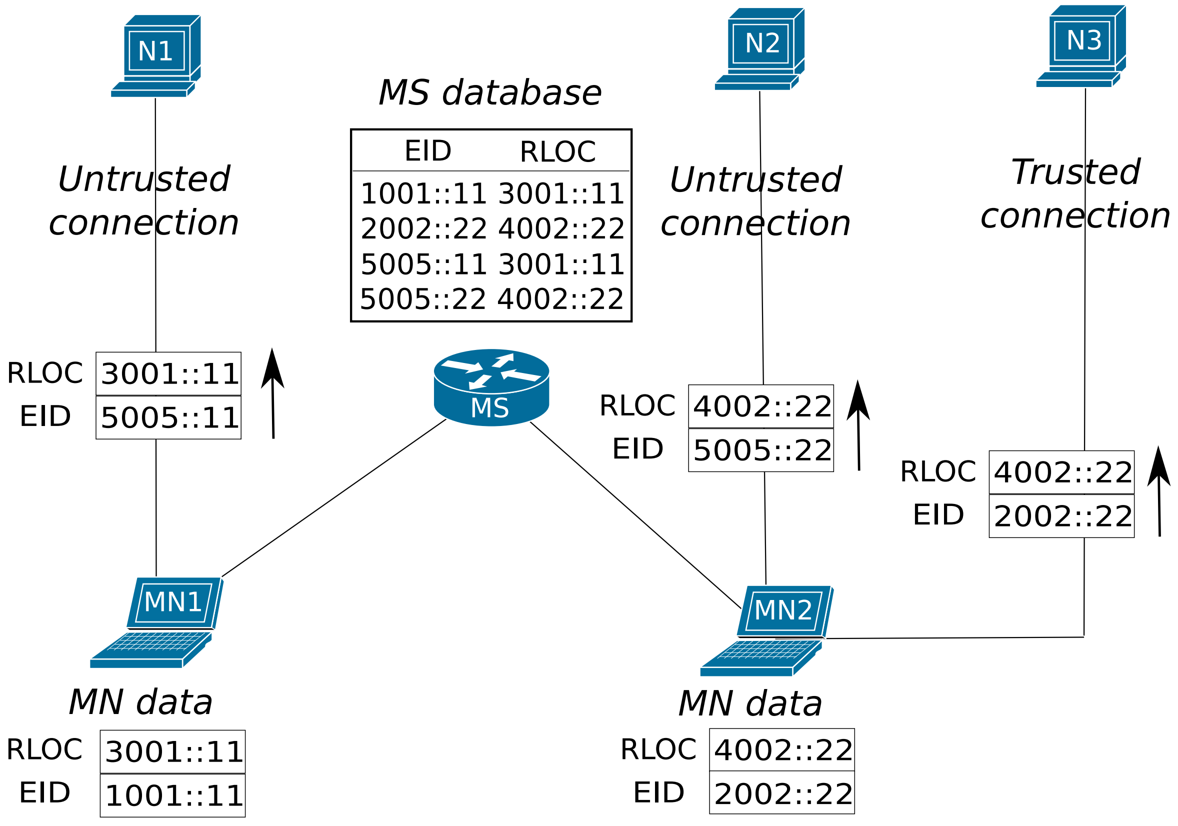
Infrastructure-less Proposal

This section describes a solution to provide MN-generated temporary EIDs (tEIDs). This solution takes advantage of the IPv6 address format and its least significant 64 bits which can be auto-configured. This idea has been (similarly) applied to plain IPv6 before (see [96] for further details). It is worth to note that this solution cannot be applied to IPv4 due to its limited address space.

The main idea behind this proposal is that a set of MNs that are sharing the same IPv6 prefix and hence, are being served by the same Map Server, can auto-generate different temporary addresses to use as EIDs. Each of these tEIDs will be under the same prefix. This way, even if an attacker can track this prefix, it cannot track individual nodes. The mechanism is more efficient as the number of MNs sharing the same prefix increases.

In order to generate the above-mentioned tEIDs, we borrow the mechanisms described in [96]. By means of a hash algorithm, the MN generates a random set of bits to fill the least significant 64 bits of a given prefix. Then the MN queries its auto-generated temporary address on its Map Server to detect duplicated addresses. If the address is already in use by another MN the Map Server replies positively and then the MN has to generate another address and query again. If the address is not in use, the Map Server replies with a negative Map-Reply and the MN knows that it can register the tEID.

In the case that two different MNs generate the same tEID at the same time, both MNs will receive negative Map-Reply at the time of checking the presence of that tEID on the Map Server, and therefore both of them will think that the tEID is available to be used. Since the probability that two different MNs generate the same tEID and query the Map Server at the same time is extremely low [94] due to the 64 bits address space, an optimistic address collision detection mechanism can be applied, i.e. the MN register its tEID as soon as it checks that it is not already registered and starts establishing connections with it. After a time-out the MN checks again the data in the Mapping System to see if its information was correctly recorded or if something went wrong during the registration process (i.e. another MN registered the same tEID). In the rare case that a collision occurred, it will roll back, drop the established connections and reinitialize the tEID registration (with a new generated tEID).



EID	RLOC
1001::11	3001::11
2002::22	4002::22
5005::11	3001::11
5005::22	4002::22

Figure 5.3 **MN generated temporary EIDs**

Figure 5.3 shows an example of the proposal. The MNs are sharing the prefix 5005::/64 to generate temporary EIDs. The last 64 bits of the addresses belonging to that prefix are generated by the MN. They register these generated addresses in the Map Server, and use them to establish connection to not trusted nodes.

With this architecture, a misbehaving node, with access to the shared prefix, could attempt to deplete the available pool of tEID addresses by registering as many as possible. Alternatively, it could also take over a tEID (and hijack its traffic) that is in use by another node, by simply registering that tEID. To avoid this, the Map Server stores a list of authorized users for each tEID prefix, while still using the existing security association (a pre-shared key for their real EID) to authenticate each individual node. Avoiding traffic hijacking can be achieved by requiring explicit dropping of a tEID in use by the previous owner.

The infrastructure-less solution can be used without additional cost in a trusted network. The nodes simply share an EID prefix for temporary address usage, and achieve identity privacy this way. This can be used by companies which own a prefix and share it between MNs of their property. If any of the MNs sharing a prefix does not belong to a domain under the company control or trustiest, then presence of misbehaved nodes should be assumed. When that is the case, there is an opportunity to sell an authentication service to the entities operating the mobile nodes. Registration is only allowed to paying customers, and a tiered service can be offered based on an anonymity quality metric defined by the provider (e.g., nodes allowed per prefix, prefix size, etc.).

Infrastructure-dependent Proposal

This approach introduces a new element, the "Anonymity Server" (AnonS). Its function is similar to that of a DHCP server [21], handing out tEIDs on demand to the MNs which request them. This AnonS can register tEIDs (update the EID-to-RLOC binding) to one or several Map Servers. The key point is that this AnonS does not register its own RLOC for the tEID, rather it registers the MN's RLOC, and hands out a lease on the use of the registered tEID to the MN. The AnonS is responsible for updating the EID-RLOC association for the tEID when necessary. The complete mechanism works as follows.

A MN wants a tEID, so it sends a request to the AnonS telling it its real EID and its current location. The AnonS stores this information and assigns a tEID from the available pool to the MN. Then the AnonS registers this tEID to the Map Server responsible for the covering prefix, with the RLOC data of the MN. When this process is completed, the AnonS notifies the MN that it can start using the tEID. When the MN wants a new address, it only has to ask the AnonS for a new one. When the MN roams, it notifies both the Map Server responsible for its real EID, and the AnonS, if a tEID is in use. Finally, the approach is secured similar to the usual

Map Server registration: authentication data is associated to each tEID request. This data is based on pre-shared keys stored both at the MN and the AnonS, and is generated as in the Map Server case (see [37]).

Figure 5.4 illustrates the solution. The AnonS keeps a database of its tEIDs (5005::55 and 7007::77) and to whom they have been assigned (5005::55 assigned to the MN1 with EID 1001:11). It also keeps record of the last known position of all the MNs using its EIDs (MN1 last RLOC is 3003::33). The AnonS tEIDs can belong to different prefixes and Map Servers (5005::55 belongs to MS2 and 7007::77 to MS3).

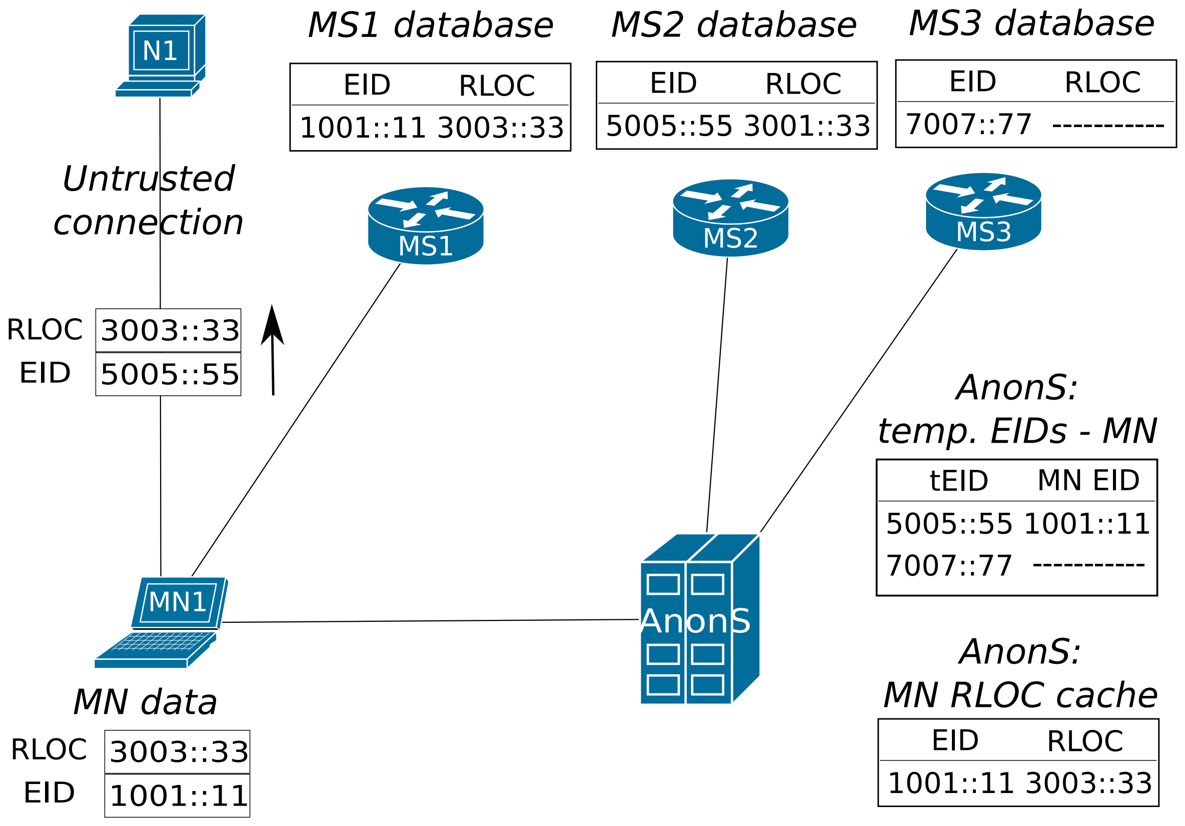


Figure 5.4 **Anonymity Server**

Deploying an AnonS generates revenue for its operator, which controls the access to the identity privacy service. At sign-up the client MN is configured in the AnonS, and a pre-shared key is stored in both entities. Pricing can be made dependent on several factors, such as the number of distinct tEIDs requested over a period, their lease time, etc. Additionally, increasing Map Server diversity by acquiring several (t)EID prefixes registered to different servers is another price differentiator, or a means to rise above competition.

5.4 Related Work

Location privacy in mobility is a well-known issue which Mobile IP has faced up in [70]. In particular they use a similar approach as the one presented in Section 5.3.1 to solve it, in this case the Home Agent acts as the proxy. The authors of [134] extend this idea by proposing to deploy redundant Home Agents to enhance privacy. Finally, a different approach to location privacy has been proposed in [118] where the authors propose to extend Mobile IP to use IPv6 ”pseudo home addresses”. These addresses are generated in a similar way the ones proposed in [96] are generated for identity privacy. Although these ”pseudo home addresses” are intended to provide location privacy, they implicitly also serve as an identity-privacy mechanism.

In general, using temporal addresses is a well-known approach to provide identity privacy in the IPv6 area, being the main proposal the one standardized in [96]. Another good example can be found in [50]. Note that [96] forces keeping one temporary identifier per connection, which can lead to runtime issues related with closing long-term connections and the maximum number of temporary addresses supported by the system. This can be observed in the current Linux kernel implementation. On the other hand, mechanisms to hand out addresses from a pool to hosts are also well-known [21]. In this thesis we have taken these established approaches and adapt them to the LISP-MN architecture.

Finally it is worth to note that in this thesis we have focused on solutions for privacy at the network level, however other approaches exist at higher layers. In this context, onion routing is one of the most well-known. Using onion routing, packets traveling through the network have been repeatedly encrypted in their origin, and are layer by layer unencrypted by the routers they go through. This way, routers in the path only know the previous and next hop of the packets. The main idea was originally proposed in [43]. A patent free, improved, and already deployed version is Tor network [18].

5.5 Analysis

This section discusses the level of security provided by the mechanisms proposed in this chapter, the trade-offs they impose and the feasibility of their implementation.

5.5.1 Location Privacy

The proxy-based approach proposed guarantees that the location of the MN is never exposed to remote nodes, however the use of a proxy introduces an inefficient routing path that degrades the performance of the LISP-MN communications. To alleviate this, the extensions proposed to improve vanilla proxy selection allow the MN to choose the most suitable RTR for its needs.

Particularly the MN can get the geo-location of the proxy and select an RTR based on that information. If that is the case, there is a trade-off on which RTR to select since closer RTRs would provide better performance, but also disclose more data about the potential location of the MN. We recommend select randomly from a set of mid-range located RTRs to balance among location disclosure and performance degradation. Besides, the load on the RTRs can be alleviated deploying more RTRs and providing their load information to MNs to help to select a non-overloaded one.

In terms of implementation, an MN compatible with NAT traversal can use the NAT traversal mechanism to get basic location privacy. The usage of the extensions proposed on this chapter requires support for parsing and encoding/decoding the extra information on both sides, i.e. on the MN and on the Mapping System. Additionally, the Mapping Systems needs to get populated with the information regarding the RTRs, how to populate the Mapping System with that information is out of the scope of this chapter.

An RTR-compatible MN requires minor implementation changes in order to use the proposed location-privacy solution. A decision process, to decide which nodes are trustworthy or not, is the only additional code to be implemented. The proposed implementation for this is the use of a configuration file in the MN, which stores a list of EIDs that the MN will trust. When receiving a query from one of those EIDs, the MN will reply with its real RLOC.

5.5.2 Identity Privacy

In this chapter we propose a simple, yet practical, design of an auto-managed identity privacy by means of auto-generated temporary EIDs that does not require of any new infrastructure deployment. The trade-offs of this approach are that it only serves for IPv6 addresses, that it imposes extra computation on the MNs and that the MNs are still traceable at prefix level. We extend it by proposing the Anonymity Severs, which enable the nodes to use identities from different prefixes, at the cost of requiring new infrastructure elements. Moreover, the use of different prefixes, gives to the users of an AnonS a higher level of anonymity than the use of traditional IPv6 privacy mechanisms. In those, the MN still can be tracked at IP prefix level, whereas with the AnonS solution the MN's EID prefix can be regularly changed among prefixes that can belong to different domains.

There is what makes the AnonS specially attractive as a mechanism to provide identity privacy and distinguishes it from the previous presented solution. The MN can use as many addresses, even from disjoint prefixes, as it wants. As a result, an attacker tracking tEIDs will have difficulties to correlate them to a single MN. An anonymity server can work with IPv4, IPv6 or both address families. In contrast to the infrastructure-less approach, using an AnonS is a viable solution for IPv4 temporary EIDs, because it optimizes address usage, in the face of the IPv4 address shortage.

Before delving into the details of the identity-privacy implementation, its common use case should be discussed. Typical users do not want (or even be aware of) privacy in their normal communications. They want to be private just when connecting to untrusted sites. Those kinds of connections are not frequent and are distributed in time. The “private mode” on modern web browsers could serve as an example of this usage pattern. With this in mind, the solution that seems more balanced between complexity and efficiency is using a single tEID rather than one per connection. This tEID is shared by all the connections that require privacy and it is refreshed after a pre-defined period. If there are active connections, then the tEID will not change until the system does not have any active (private) connections. The amount of tEIDs required to provide a unique one to each connection can be potentially huge. Having just one tEID changing over time keeps the complexity of the implementation at a reasonable level and is enough to fulfill the requirements of the common use case.

Another issue is how the system decides which connections require identity privacy. Leaving this to the network-layer is not trivial, since it does not usually have enough information. The proposed approach is to delegate this decision to the upper layers. Each application decides which connections use the tEID (for instance as the private browsing mode). In order to implement this, we propose using a new socket option [97]. This provides the programmers the flexibility to choose when privacy extensions should be applied. For backwards compatibility with existing applications not using this socket option, an alternative is proposed by means of a connection-manager application. The connection-manager can be used to enable or disable identity privacy globally, for all applications, by switching between the real and temporary EIDs.

5.6 Evaluation

This section evaluates the proposed solutions to asses the extent of their impact in both the data and control planes.

5.6.1 Data-Plane

In terms of evaluating the data-plane burden imposed by the proposed solutions, it is worth to note that both approaches for identity privacy do not modify the data-plane operation and therefore do not impose any burden. On the other hand, the solution for location privacy does impose a penalty in the data-plane since, like in every proxy solution, the use of another

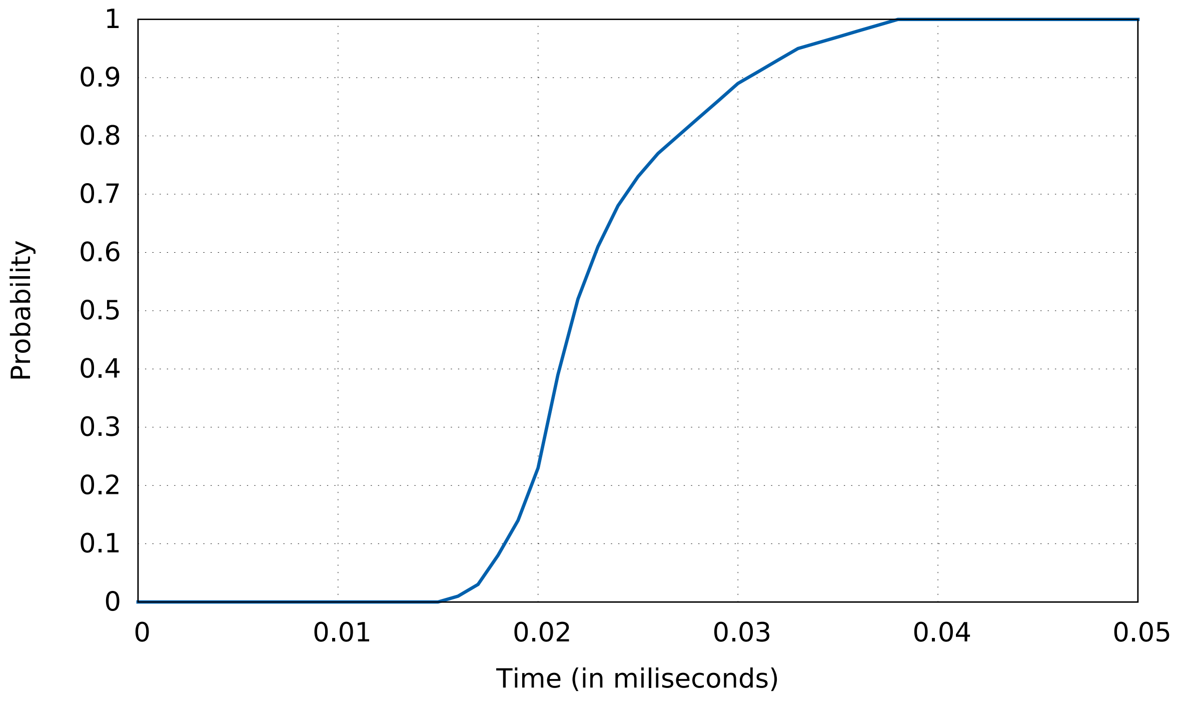


Figure 5.5 **Constant delay introduced by the RTR**

element in the path increases end-to-end latency due to the longer path and the extra processing time. The latency increment caused by the path stretch varies depending on the location of the RTR chosen, however the processing time at the RTR is constant. To evaluate how much constant delay an RTR introduces, we have built a prototype of the proposal in 5.5.1 using the OpenOverlayRouter open-source implementation [100] following the topology depicted in figure 5.2.

Figure 5.5 shows the Cumulative Distribution Function of the processing time of packets (1000000 packets at 1000 packets per second) at an RTR running OpenOverlayRouter on a desktop Linux machine (2GHz dual-core with 1 GB of RAM). The figure shows that most packets suffer a delay of less than 30 microseconds, which is negligible for most scenarios. Furthermore, it is expected that hardware-based LISP solutions [81] can provide even lower processing times.

5.6.2 Control-Plane

We evaluate the control-plane modifications in terms of the number of extra control messages that are required to exchange between entities in order to support the proposed solutions. For location privacy, there is no need for any extra messages, since the signaling is the same that is used to perform NAT traversal. The identity-privacy solutions require however additional control messages.

The infrastructure-less approach requires no extra control messages if there is no duplicate address detection or collision check. If the MN looks for duplicated addresses, then one request to the Mapping System and its reply are needed. Twice this number if collision check is also performed. The infrastructure-dependent solution doubles the number of signaling messages of vanilla LISP registration due that first it is the MN who registers to the AnonS, and then is the AnonS who registers to the Mapping System. Note that in this case the MN does not register the tEID to the Mapping System since this is done by the AnonS. Table 5.1 summarizes the extra signaling messages required between entities.

	MN to MS	MN to AnonS	AnonS to MS
Location (with NAT capable nodes)	0	0	0
Identity: Infrastructure-less (no check for duplicates/collisions)	0	0	0
Identity: Infrastructure-less (checking duplicates)	2	0	0
Identity: Infrastructure-less (checking duplicates & collisions)	4	0	0
Identity Infrastructure-dependant	0	2	2

Table 5.1 **Extra control messages required between entities**

5.7 Conclusions

In this chapter we have presented a set of solutions to provide location and identity privacy to LISP Mobile Nodes. Location privacy is a well-known problem usually solved by proxies. Here we have presented a proxy based solution that takes advantage of the LISP NAT-traversal mechanism and extend it to better serve the location privacy purpose.

We have also proposed two different approaches to solve the identity privacy issue. Based on the idea of using temporary identifiers to hide the real identity of the MNs we have defined different solutions adapted to different scenarios. The first approach does not require (or requires just a few) modifications to the LISP infrastructure, it is based on temporary auto-generated identifiers. The second one requires the deployment of a new element called Anonymity Server.

It serves as a kind of DHCP server to provide and manage heterogeneous and distributed temporary identifiers.

We have addressed both privacy issues taking a realistic approach aiming for deployment. In particular we have briefly discussed the trade-offs of the proposed solutions alongside with the feasibility of their implementation. The evaluation shows that the burden that the solutions impose in the data and control plane operations is reasonable.

Chapter 6

OpenOverlayRouter: Architecture and Evaluation

OpenOverlayRouter (OOR) is an open-source implementation to deploy programmable overlay networks. It leverages on LISP as the overlay control protocol to map identifiers in the overlay to locators in the underlay. It implements both LISP and LISP-MN specifications for operational state exchange and, besides, has support for NETCONF management and VXLAN-GPE encapsulation. OOR is a multiplatform implementation with current support for Linux, Android and OpenWrt. It targets flexibility, portability and extensibility by taking a user-space approach for its implementation. In this context OOR represents a solid base for research, innovation and prototyping of new overlay use-cases and applications. It is particularly useful to deploy programmable overlays at end-nodes in order to provision them with SDN capabilities as described in Chapter 3. This chapter describes the OOR software architecture as well as how it overcomes the challenges associated with a full LISP user-space implementation. Finally, this chapter presents an experimental evaluation of its performance in relevant scenarios.

6.1 Introduction

In this chapter we present the OpenOverlayRouter (OOR) open-source project, a community-driven LISP-based overlay implementation for PCs running Linux [79], home routers with OpenWrt [107] and Android devices [2]. Open-sourced under an Apache 2.0 license [3], OOR supports both LISP [30] and LISP-MN [32], a lightweight version of LISP intended for mobile devices. Furthermore, OOR can be remotely managed and configured via the NETCONF protocol [24] and can leverage on the VXLAN-GPE [72] encapsulation format to encapsulate overlay traffic trough the underlay. OOR is a renaming of the LISPmob project [82], the original implementation of the LISP-MN specification. The LISPmob implementation grew in features and capabilities over the years. From a minimal LISP-MN implementation it evolved into a full LISP implementation, and from there to a complete overlay solution that incorporates other protocols beyond LISP.

Despite there are other LISP implementations, both proprietary (Cisco LISP [81]) and open-source (OpenLISP [113] for BSD systems), OOR has a completely different focus and scope. OOR's code runs entirely at the user-space and has a common code-base for all its supported platforms. This software approach allows the OOR community to have a strong focus on flexibility and customization. Indeed, OOR's architecture represents a solid base for research, innovation and prototyping of new LISP use-cases and applications. A good example of OOR's success is that it is being used by the LISP project in the OpenDayLight [99] SDN controller. Additionally, the OOR community has developed *liblisp*, a standard library to parse LISP messages that simplifies the development and interoperability between different LISP entities/implementations.

OOR is particularly useful to enable an SDN deployment in the end-node scenario described in Chapter 3. End-nodes to be provisioned with SDN capabilities are likely to require an overlay-oriented data-plane (e.g. VXLAN-GPE) and a pull-based connectionless southbound protocol (e.g. LISP). Such end-nodes can use OOR to fulfill their SDN requirements while leveraging on its support for typical end-node platforms (i.e. Linux, Android, OpenWrt). Furthermore, the homogenous implementation across platforms (thanks to OOR user-space approach) eases the deployment of an SDN infrastructure over a wide range of heterogeneous end devices. In this sense, OOR enables the deployment of overlays (like the one depicted in Fig. 6.1) that hide the inherent heterogeneity of the underlay and offer an homogenous view of the overlay network suitable to be programmed via SDN.

In what follows we describe the OOR software architecture and components, with focus on the challenges that the community has overcome to implement a full multi-platform LISP user-space implementation. We first give an overview of the main architectural core ideas behind OOR implementation to latter delve into the internals of its software architecture. The analysis of the software architecture is split into data-plane and control-plane. In the data-plane section we describe how OOR encapsulates traffic into LISP data packets and how it decapsulates traffic from LISP data packets. In the control-plane part, we show how LISP control is managed, both in terms of control packets processing and control state handling. Both data-plane and control-plane are implemented in a single modular user-space daemon. Finally, we present an experimental evaluation of OOR in relevant scenarios and compare its performance with other related implementations. As the results show, despite taking a user-space approach, OOR implementation results in a remarkable performance suitable for home and edge devices.

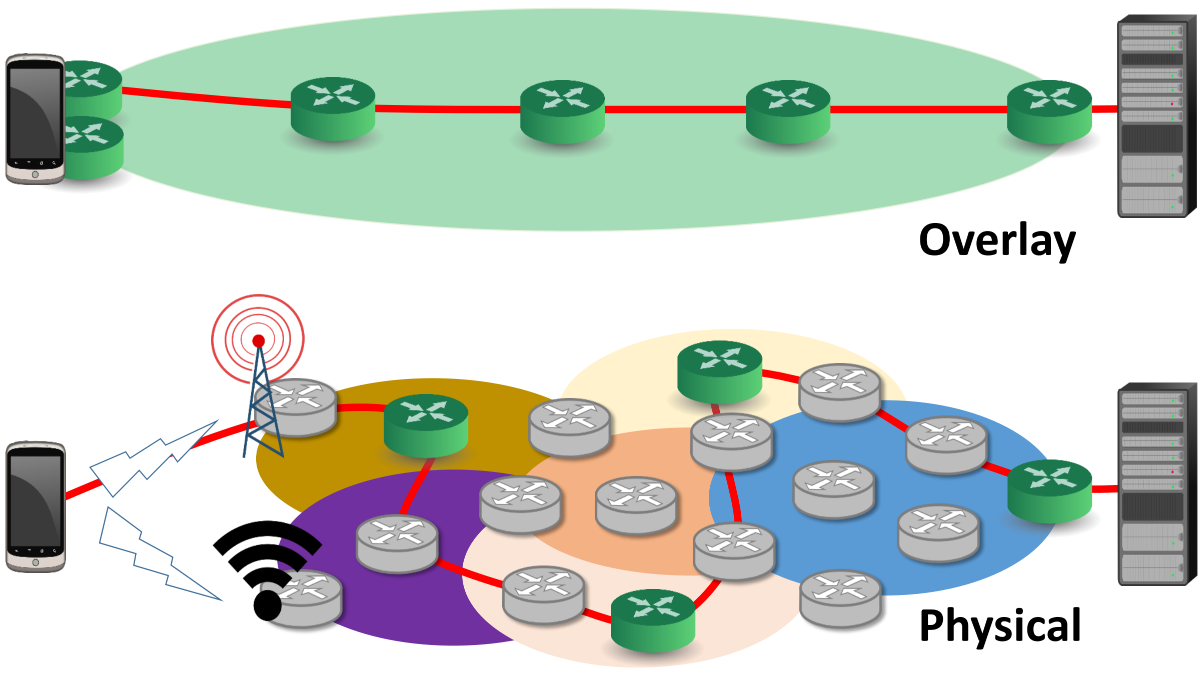


Figure 6.1 **Overlay approach**

6.2 Architecture Overview

OOR is an open-source implementation of both LISP [30] (see Section 2.2) and LISP-MN [32] (see Section 5.2) developed in C [65] for Linux-flavored systems. The main goal of OOR is to represent a solid code-base for overlay research, innovation and prototyping with an end-user focus. To achieve this, OOR comprises a modular architecture with a user-space approach and a multi-platform implementation.

6.2.1 Modular Design

OOR is composed of different software modules. The modules have been abstracted and its 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 figure 6.2. 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 the control state, handles control signaling and manages the mapping information that is used by the *data* module. OOR implements several LISP devices, each of these devices is represented with a module that adapts the *control* and *data* modules behavior to match the specific LISP device. Beyond those main modules, there are certain parts of OOR that have been abstracted into auxiliary modules that connect to the main ones, such as the interface management, the multihoming procedures or the database storage. Finally, OOR implements and makes publicly available the *liblisp* library to parse LISP packets.

6.2.2 User-Space

All OOR code runs in user-space, this prevents having to delve into hardware specific optimizations while avoiding the complexity and high maintenance costs typically associated with kernel code. To support a LISP data-plane on user-space, OOR uses TUN/TAP [138] 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 6.2 depicts OOR components in user-space and how they interact with the kernel space. The *data* module hooks to the TUN interface for data processing while the *control* module opens a socket on the WAN interface for control signaling. Finally, OOR uses netlink to monitor and modify the routing tables. With this approach, OOR does not need specific support at the kernel and can be easily ported to different Linux-flavored systems.

6.2.3 Multi-Platform

Since its inception OOR has been oriented towards end-users. In order to do so, it tries to provide an easy usage and configuration, while at the same time offer LISP features that represent immediate advantage, such seamless mobility or bandwidth aggregation. However, to deliver such benefits to a broader set of end-users, it is mandatory to offer support for different platforms. Thanks to its modular architecture and its user-space approach, at the time of this writing OOR supports three different platforms using the same code base. Taking base on an implementation for Linux, OOR has been ported to OpenWrt (home routers) and Android (mobile devices). Although with some platform-specific caveats (see Section 6.4.4), most of the benefits that OOR offers are available for all the three platforms.

6.3 Control-Plane

The LISP control-plane registers, stores, and resolves EID to RLOC mapping information. This section describes the relevant implementation details of the OOR control-plane, mainly implemented by the *control* module.

LISP control signaling is send and received by means of opening a socket in the LISP control port in the WAN interface. The way it implements control-plane, which control packets it sends and how it reacts to received control packets is governed by the LISP device that is configured. See details on this on 6.3.1. To keep track of periodic events, the *control* module

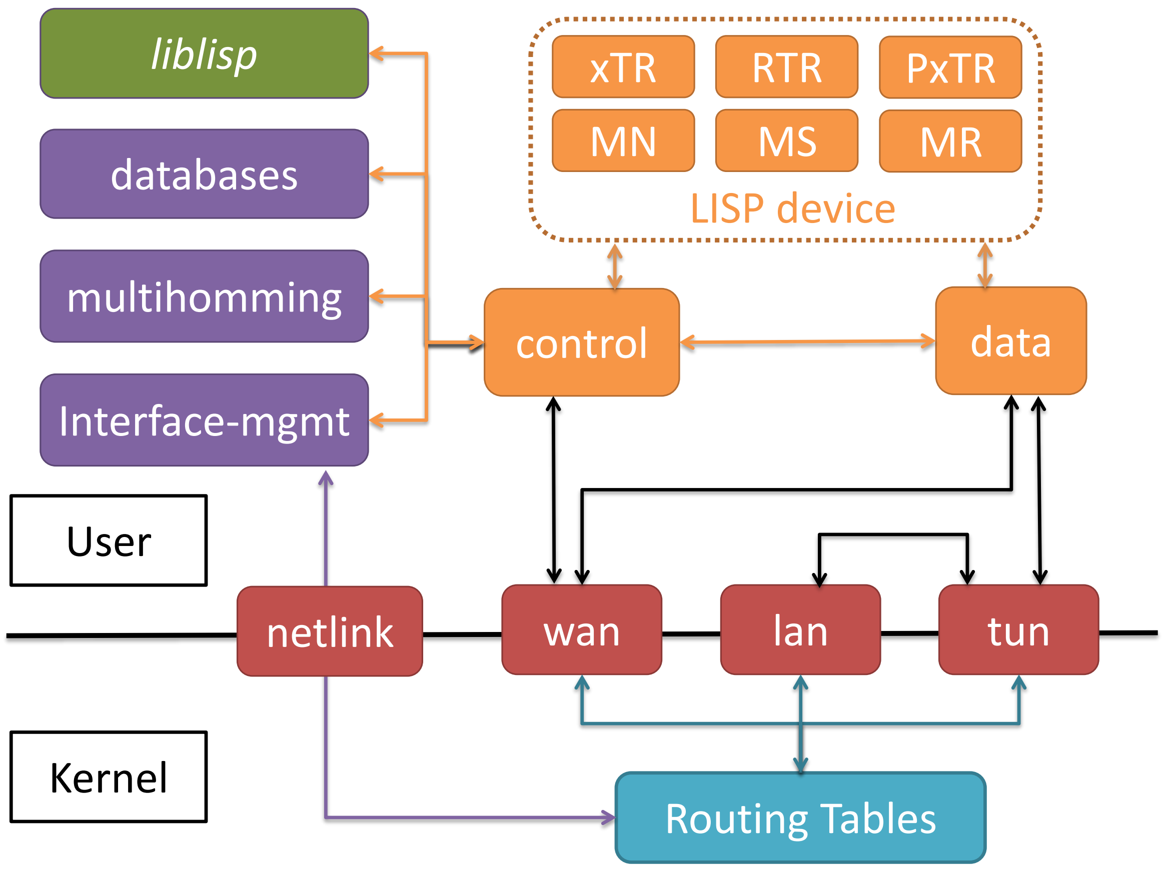


Figure 6.2 **OOR architecture**

relies on a set of timers, see Section 6.3.3. The local and remote mapping information are stored in two different databases that are described in Section 6.3.6. The remote configuration support via NETCONF is described in 6.3.5. Finally, the multihoming features are presented in 6.3.7 while the interface management is described in Section 6.3.8.

6.3.1 Devices

OOR supports different LISP devices: Ingress/Egress Tunnel Router (xTR) [30], LISP Mobile Node (LISP-MN) [32], Map-Server (MS) and Map-Resolver (MR) [37], Proxy Ingress/Egress Tunnel Router (PxTR) [77] and Re-encapsulation Tunnel Router (RTR) [31]. This is achieved through a unified control-plane module that supports pluggable device-specific modules that govern the way the LISP control-plane is executed, that is, how the control-plane reacts to an incoming LISP control message or to an expired timer.

The LISP data-plane is similar regardless the LISP device, however the control-plane is device specific. OOR control module is governed by the specifics of the LISP device that has been configured, which means that the control module will react in a different way to an incoming control message or to an expired timer depending on the LISP device OOR is running as.

6.3.2 Control Signaling

LISP control messages are (generally) not LISP encapsulated. OOR sends these packets through a physical output interface. In order to receive LISP control packets from other LISP entities, OOR opens sockets on the LISP control port. When a packet reaches one of these sockets, OOR triggers the control message processing mechanism. How OOR reacts to incoming control packets, and which outgoing control packets it sends, depends of the specific LISP device OOR is behaving as. OOR will perform different actions, depending on the type of the packet received and on which LISP device OOR is acting as. For example, when OOR behaves as a xTR in the case of a Map-Reply, it will store the new arrived mapping on the *map-cache database*.

6.3.3 Timers

LISP protocol requires performing some actions periodically, for instance when operating as an xTR it has to send control messages to the Mapping System periodically. Besides, some of the information stored is only valid for a certain period, after that it is considered outdated and it is discarded. OOR keeps track of periodic events using a timer wheel algorithm, similar to the one the Linux kernel uses for its internal timers. Event timers are placed in a wheel structure that rotates over time. Its position on the wheel indicates when the timer is going to expire. Timers are created with an assigned callback function and related data. Whenever the timer expires, the callback function is called with the associated data as a parameter.

6.3.4 liblisp

OOR implementation provides the *liblisp* library to help parse LISP control messages. The library allows reading raw LISP messages to store them into internal buffers as well as to write internal buffers into raw messages. The buffers are accessible through a set of well-defined structures and functions. This is specially useful to handle packets containing addresses in the LISP Canonical Address Format (LCAF) [33], since *liblisp* abstracts all the complexity. The *liblisp* library is independent from OOR and thus can be used by any LISP implementation or entity to simplify its development or interoperability.

6.3.5 NETCONF

The main way to configure OOR is via a static configuration file that is read during the boot-up process. This file contains information required for the correct operation of the LISP protocol as well as user-configurable aspects of OOR, e.g. which encapsulation format to use. During run-time it is possible to modify parts of this configuration remotely thanks to the NETCONF [24] support that OOR implements. NETCONF support at OOR is based in the *libnetconf* library [78, 73] which offers an open-source C implementation of the NETCONF protocol. OOR implements support for a simplified version of the YANG [6] datamodel for LISP [26] that allows, for instance, to change the Map-Resolver in use or to modify the EID prefixes currently configured in the local mapping database (see Section 6.3.6).

6.3.6 Databases

OOR stores mappings in two different data structures, depending on whether these mappings are local or not. The mappings related to EIDs prefixes belonging to the OOR node itself are stored in the *mapping database*. Mappings related to EIDs belonging to remote nodes are stored in the *map-cache*. The *mapping database* is loaded from a configuration file during boot-up and remains mostly static during OOR execution (unless modified via NETCONF as described in Section 6.3.5). As an example, when running as an xTR this database stores the EIDs that must be encapsulated and when operating as a Map-Server it stores the EIDs that OOR is serving. On the other hand the *map-cache* is dynamically populated during OOR operation, i.e. on a *map-cache* miss triggered by the data-plane, the *control* module retrieves the missing mapping from the Mapping System and stores it on the *map-cache*.

Despite serving different purposes and being populated in different ways, the two databases share a similar internal structure. Both are implemented over a Patricia Trie structure (also known as Radix Tree) that is kept in memory. A Patricia Trie is an optimized version of a digital tree where single-child nodes are merged with their parents. This produces an optimal data storage for strings that share long prefixes, such 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 to retrieve the most specific prefix for a given IP address. OOR's modularity supports as well using other databases for non-IP based mappings.

6.3.7 Multihoming

LISP inherently supports multihoming, a LISP mapping can bind a single EID prefix to several RLOCs and hence, physical interfaces. In the EID-to-RLOC bindings each RLOC may have its own priority and weight, the priority indicates the preference to use a given locator while the weight describes how the traffic should be balanced in case that several locators have the same priority. Since LISP sites register their mappings in the publicly accessible Mapping System, the priority and weight values express the inbound traffic engineering policies.

In multihoming scenarios -where several locators are available at the same time- the user has to define the inbound and outbound traffic policies, this is done by configuring priorities and weights for the available locators. 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 is done to avoid splitting flows over different paths that may have different delay/jitter and hence, may severely impact the performance.

To balance load among several locators with the same priority, OOR uses weighted vectors containing locators. Each mapping (i.e. each EID prefix with a set of locators) has its own weighted locators vectors. The locators sharing the maximum priority a mapping are distributed along the mapping vectors according to their weight. For instance, if two locators have weights 10 and 15 the multihoming module will assign the 40% of the vector positions to the first one and the 60% to the second. When OOR needs to select a locator to encapsulate to the EID it chooses one from these vectors. To select the proper locator, it uses a hash function to haphazardly select a position of the vector. The locator pointed by that vector position will be the one used. OOR keeps three vectors per locator set, one containing the IPv4 locators with maximum priority, another the maximum priority IPv6 locators and a third one with both IPv4 and IPv6 locators if they share the maximum priority. One or more of these vectors may be empty or be equal to another vector. This is the case when there are no locators for one type, or when the maximum priority of one type of locators is not equal to the other type.

OOR uses this selection process to select locators to use both for destination address (belonging to remote site) and source address (belonging to OOR). Since OOR can use both IPv4 and IPv6 locators, it checks first the IP version of locators available on both sides. If they only have locators from different version, no communication is possible (except by means of a proxy). If they have locators from at least one common version, the selection process is performed taking into account just that IP version. If both remote site and OOR have IPv4 and IPv6 locators, all locators are taking into account during the selection process. However, source locators are selected first. That means that if in source locator selection, an IPv6 locator was selected, during destination locator selection only IPv6 locators will be taken into account. The locator balancing vectors are calculated in advance for both OOR local locators and remote sites locators, and kept in memory to be checked when needed.

However, it should be noted that in OOR there is a limitation of just one IPv4 locator and one IPv6 locator per interface (not taking into account IPv6 link-local addresses). For instance, if there are two IPv6 global addresses configured in one interface, OOR will use just one of them. OOR handles its inbound traffic policies by registering these priorities and weights on the Mapping System.

Users can take advantage of OOR multihoming capabilities. First, they can deploy active-backup multihoming solutions for disaster recovery scenarios. Second they can make the most of active-active multihoming deployments thanks to the traffic balancing across the links that entitles them to perform bandwidth aggregation.

6.3.8 Interface Management

The interface management is critical specially in two scenarios, mobility and multihoming. To support mobility, OOR has to detect handover from one network attachment point to another. On multihoming scenarios it needs to detect any change on the interfaces to adjust the announced locators and the load balance vectors. These two scenarios apply to both xTR and MN modes, as a result OOR transparently supports multihomed MNs or mobile routers. Interface management is of key importance in OOR given that it has to seamlessly support multihoming and handovers in different platforms and Linux flavors.

In order to manage the system interfaces OOR opens a netlink socket to the kernel, this is used both to modify the routing tables as described in Section 6.4 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 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 reacts by updating its internal structures and, if needed, the Mapping System information as well as the remote LISP peers through control signaling. This control message tells them that there has been a change in the mapping information regarding OOR node and that they should update their map-caches. In case of an interface going up or down it follows the same procedure, but it 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 needed, it updates the routing tables to handle outgoing RLOC packets (see Section 6.4).

6.4 Data-Plane

This section presents how OOR implements the data-plane of the different LISP devices as well as the challenges of implementing a user-space LISP data-plane on different platforms. The OOR data-plane is implemented by the *data* module, which is responsible of encapsulating and decapsulating data packets. Although some LISP devices do not require data-plane (Map-Server/Map-Resolver), for the remaining devices (xTR, RTR and PxTR) its operation is quite similar, only Mobile Nodes (MN) require a slightly different approach. As in the case of the control-plane, the LISP device modules modify according to their particular needs the *data* module operation. Mostly due to its condition of user-space implementation, OOR data-plane has faced some implementation challenges. The specific encapsulation format to be used can be chosen prior to boot-up OOR via a variable in the configuration file. At the time of this writing, OOR implements support for both LISP [30] and VxLAN-GPE [72] encapsulation formats.

6.4.1 Tunnel Routers (xTR, RTR, PxTR)

OOR implements the data-plane for all LISP tunnel router devices (xTR, RTR, PxTR) with a similar approach. This section describes the data-plane as for an xTR, pointing out the specific details of RTR and PxTR data-planes when needed.

For 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 the outgoing EID traffic (both on xTR and PxTR modes) OOR redirects it to the TUN interface and retrieves it. The redirection is achieved by means of 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 EID space outgoing traffic 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.

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

In multihomed scenarios with several default routes OOR must ensure that Linux chooses the appropriate outbound interface. In order 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 standard UDP socket listening for LISP data traffic. Received RLOC traffic is decapsulated and written in the TUN interface, then the kernel forwards the EID packet to the EID space. In RTR mode the traffic is re-encapsulated with new RLOC headers and forwarded back to the RLOC space.

6.4.2 Mobile Node (MN)

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 the packets from an external source, while in a LISP mobile node (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. The applications running on the MN bind sockets to that interface and hence they use as a source address the EID. 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 also specific tables and rules per each RLOC interface and therefore, once encapsulated, traffic will be forwarded to the correct outgoing interface preventing loops.

On reception in Linux, OOR deactivates reverse path forwarding (RPF) verifying mechanisms to prevent discarded 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 packets arriving via a physical interface would not pass the RPF check.

6.4.3 IPv6 Encapsulation

OOR is agnostic to the address family of the underlay. It supports both IPv4 and IPv6, which serves to overcome any transition scenario. Users can deploy OOR to get IPv6 access over a IPv4 only connection, or to get IPv4 over IPv6. However, receiving encapsulated data packets on user-space using IPv6 sockets requires a different approach than for IPv4. Both LISP and VxLAN-GPE are UDP encapsulated protocols and their specifications state that the UDP checksum of the encapsulation header should be zero, however the IPv6 standard specifies that the UDP checksum must be computed.

As a result, the Linux kernel automatically drops incoming IPv6+UDP packets with zero checksum. To overcome this issue OOR uses raw UDP input sockets instead of common UDP binded-to-port sockets therefore OOR captures all UDP packets and processes only the ones addressed to the LISP or VxLAN-GPE port. This approach allows OOR to receive IPv6 zero-checksum UDP packets.

Additionally, the encapsulation and decapsulation operations require access to low level details of the IP headers such as the TTL or ToS value. Outgoing packets are received with full IP headers at the TUN interface, however incoming packets require a different approach. For IPv4, input raw sockets provide all the required IPv4 header fields at the user-space however IPv6 raw sockets only provide the IP payload. In order to obtain the IPv6 header information OOR uses IPv6 specific socket options: IPV6_HOPLIMIT to get IPv6 TTL value and IPV6_TCLASS to get the ToS value.

6.4.4 Non-rooted Android

The key challenge when implementing LISP in Android is that, as shown in previous sections, OOR requires root access because -for instance- it makes use of raw sockets. Since this requirement strongly limits the deployability of LISP on smartphones, to overcome this issue the OOR community has developed a non-rooted Android application. On that scenario some root-only features are not available, and thus some workarounds are needed.

The lack of root access is partially alleviated by the use of the Android VPN API[141] (present from Android 4.0 onwards). The VPN API must be used from a Java application. On the Android version of OOR, there is a wrapper application written on Java that calls the native OOR C application through JNI (Java Native Interface) [62]. VPN API allows applications running without root privileges to create a TUN interface, this interface is managed by the Android VPN service itself rather than by the application. The interface creation requires of explicit user permission (i.e. a pop-up is shown in the device display). Android VPN API does not allow the fine grain forwarding achieved with root access, however it does allow setting a socket as *protected*. The traffic sent through a *protected* socket will ignore the TUN interface. OOR takes advantage of this to send traffic once it has been LISP encapsulated, the system will automatically select the outgoing interface among the available physical interfaces and will use the address assigned to that interface as the source RLOC address for the encapsulated traffic.

Since on non-rooted devices there is no access to the routing tables and rules, neither to the source address of the encapsulated packets, multihoming is not available on those devices. There are not raw sockets available either, which means that incoming IPv6 traffic can not be directly retrieved. As discussed in Section 6.4.3, if an IPv6 packet arrives with zero checksum, the system will drop it. At the time of this writing, most of the current LISP implementations, such as those deployed at the beta-network [80], do not compute the outer header UDP checksum when encapsulating packets into LISP. In practice this implies that IPv6 RLOCs functionality on non-rooted Android devices is very limited.

6.5 Evaluation

This section presents an experimental evaluation of the performance of OOR. At 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. Instead, and when relevant, we compare the OOR performance with OpenLISP [113], a BSD kernel LISP implementation and OpenVPN [102], a well-known VPN software that also encapsulates packets at the user-space.

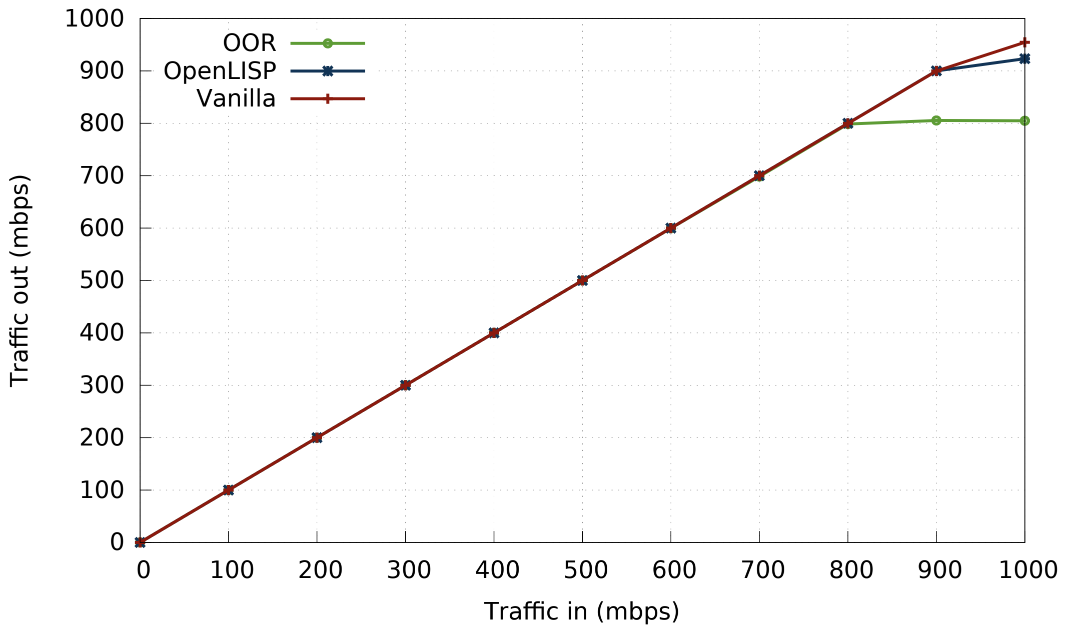


Figure 6.3 **Throughput (Linux)**

6.5.1 Throughput

Here we focus on the throughput of the OOR 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. The traffic is generated using the *nuttcp* tool (UDP packets of 1388 bytes) and we monitor both input and ouput rate.

As the figure 6.3 shows, OOR scales close to the link capacity with a maximum throughput of 800Mbps, at this rate the user-space OOR process is using all the available CPU. OpenLISP with its kernel implementation is very close to the link capacity.

We also compare the throughput of the Android and OpenWrt OOR versions 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. For Android we generate traffic using *iperf* running on Nexus 7 (Android 4.3) over a WiFi link (802.11g), for OpenWrt we run OOR on a Netgear home router (WNDR3800, OpenWrt 12.09) and we generate traffic with *nuttcp*. As shown in figure 6.4 OOR outperforms OpenVPN in both cases. Both OOR and OpenVPN show a decrement of their performance under high-loads, this is because the CPU is also used when receiving packets.

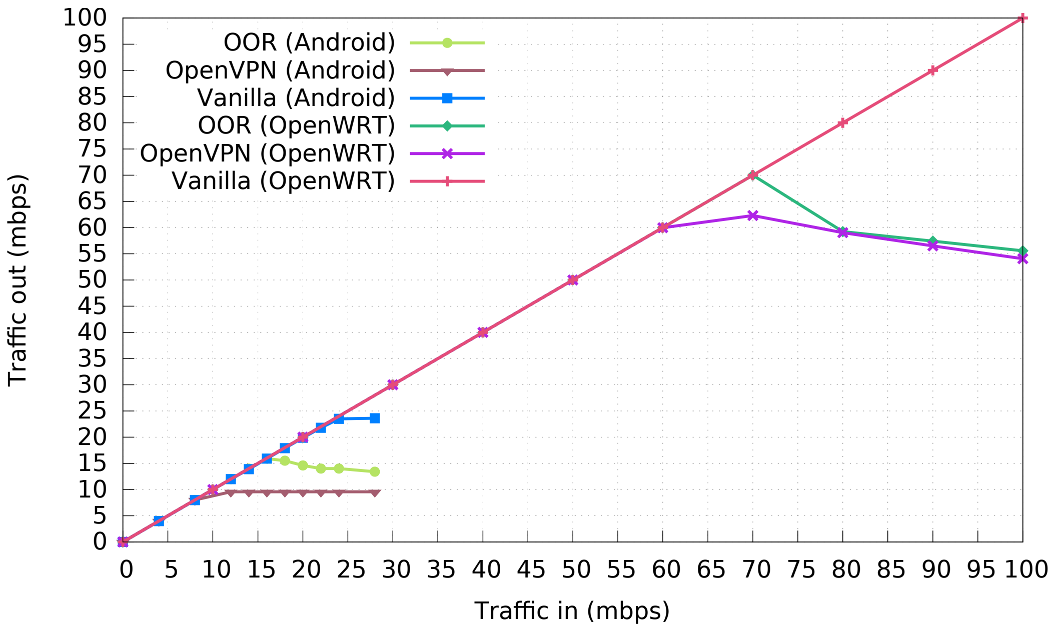


Figure 6.4 **Throughput (Android and OpenWrt)**

6.5.2 Multihoming

As described in Section 6.3.7 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 figure 6.5 shows OOR dynamically takes advantage of the available interfaces resulting in an efficient multihoming solution, where the overall throughput is limited by the overhead introduced by the different headers, including the LISP encapsulation.

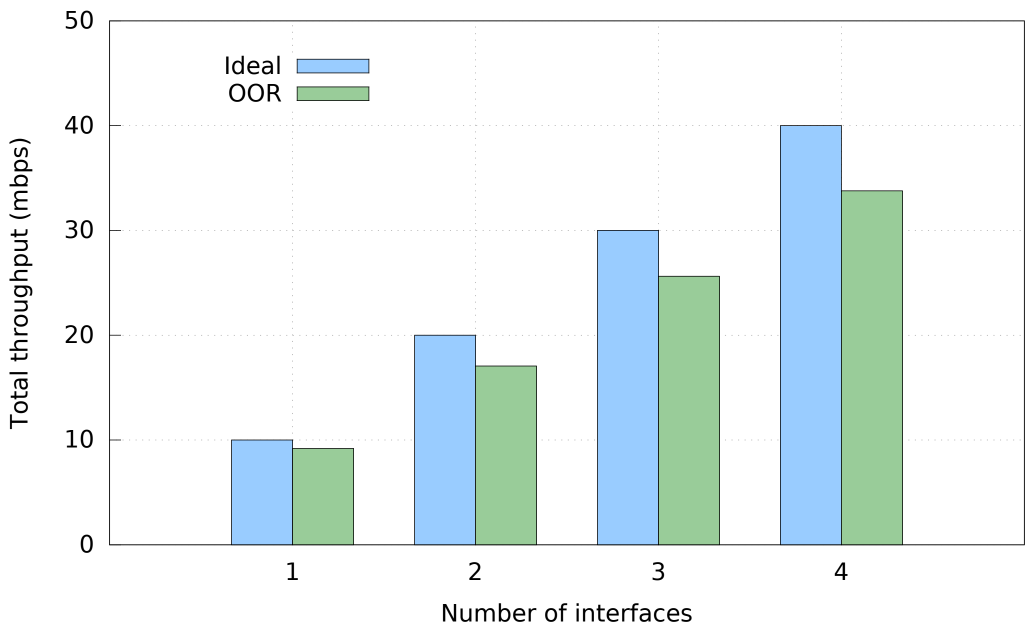


Figure 6.5 **Multihoming performance**

6.5.3 Horizontal Handover Latency

In this section we focus on the handover latency of the OOR Android version. We run it on a Nexus 7 tablet and 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 where the host is not receiving packets.

Figure 6.6 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 two scenarios, indeed the average WiFi to 3G handover latency measured over 31 tests is 4.16ms and 3.98ms for OOR 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 connectivity via WiFi.

6.5.4 Data-plane Processing Latency

Finally in this section we measure the processing latency of the OOR data-plane as a function of the size of the map-cache, specifically the time since OOR reads a packet from the TUN interface until that packet has been encapsulated and forwarded. In order to populate the map cache we use a 450k entries BGP table from the Route Views project [128] and consider them as EID prefixes, with this we simulate an xTR operating as border router with a realistic RIB. Furthermore, we split this table in smaller chunks and use them to gradually increase the map-

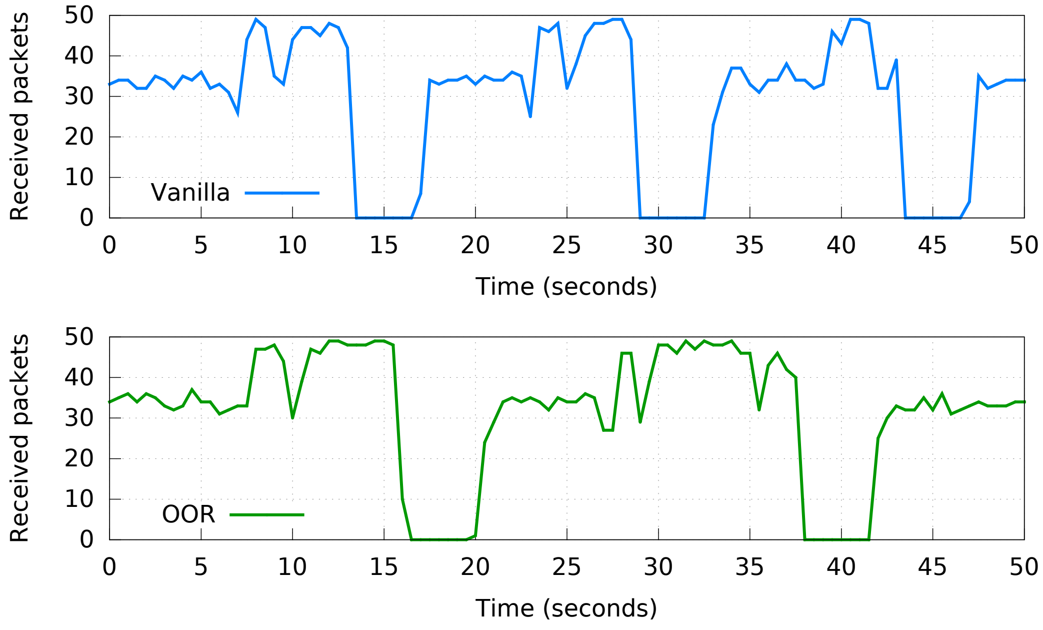


Figure 6.6 **Handover time**

cache size and assess the processing latency for different map-cache sizes. Our experiments show that OOR has an average processing latency of 40μs (as depicted in Fig. 6.7) and that this latency is independent of the size of the map-cache, validating the performance in terms of latency of the implementation.

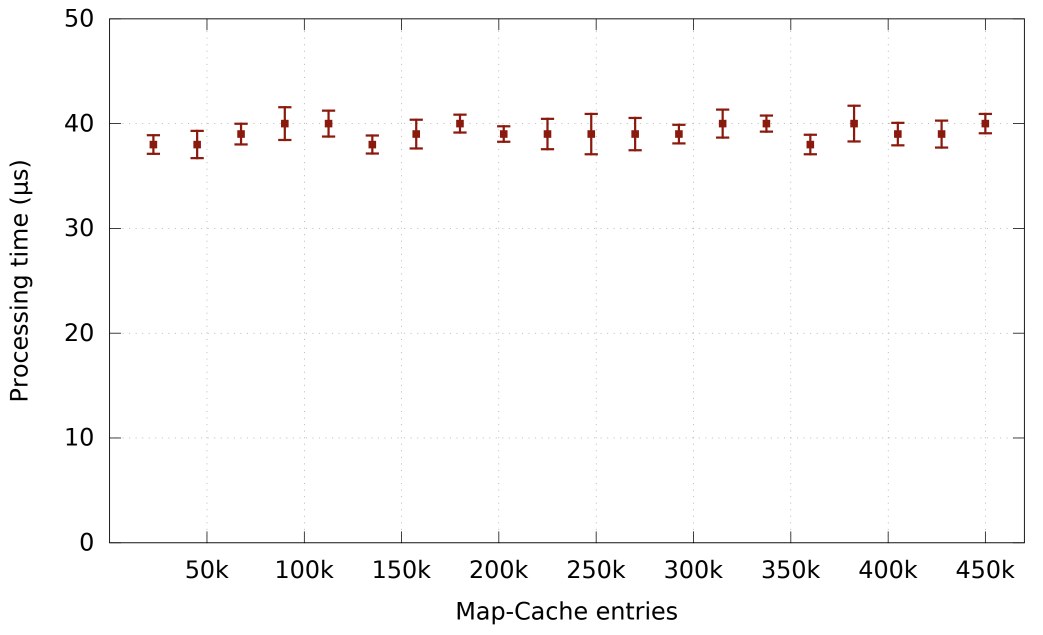


Figure 6.7 **Data-plane processing latency**

6.6 Conclusions

The OOR project provides a mature user-space LISP implementation for research, innovation and prototyping. The code has been architected with this in mind, providing an extensible and flexible LISP implementation while keeping low complexity and easy deployment. The feedback received from the community as well as the diverse projects where OOR's code is being used strongly supports these claims.

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

Chapter 7

Conclusions

7.1 Thesis Summary

The main contribution of this thesis is the concept of decoupling state and control in SDN networks. This architectural distinction reduces the complexity of controller design and allows to face the scalability challenges of each element (i.e. control and state) individually. In the first part of this thesis we have analyzed LISP as a southbound protocol suitable to be used for state exchange in decoupled-state architectures. Then, partially leveraging on LISP, this thesis has proposed two different decoupled-state SDN architectures for two specific use-cases.

On one hand, we have introduced an asynchronous decoupled-state architecture to enable SDN for end-nodes. It leverages on a distributed state database that offers a connectionless pull-based interface to southbound nodes. The state database is then connected to a decentralized and symmetric controller that exposes an intent-based interface to northbound applications. The abstract northbound polices are rendered by the controller into specific state and stored in the state database. Southbound nodes retrieve on demand state directly from the state database. On the other hand, this thesis has described a decentralized SDN architecture designed to support NFV on operator networks. This proposal decentralizes the controller instances and pushes them close to the data-plane devices they control. The state is kept in a centralized and global database that is shared among all the controllers to enable state federation and global coordination.

In the second part of this thesis we have gone in deep into the requirements and characteristics of SDN for end-nodes. Particularly, we have described the LISP protocol when applied to mobility scenarios and discussed location and identity privacy enhancements for LISP mobile end-nodes. To conclude the thesis we have presented the OpenOverlayRouter project, an open-source implementation to deploy LISP-based programmable overlays.

7.2 Open Research

This work has introduced decoupled-state architectures, a new sub-field within SDN research that to the best of our knowledge has never been discussed before. This thesis shows the motivation behind the idea of decoupling state and control and remarks the benefits of applying decoupled-state architectures to certain use-cases and scenarios. However, this new area present a set of new research challenges that can be extracted from the work discussed in this thesis. In this section we highlight open research on the field of decoupled-state architectures and the future work to be addressed in order to accomplish the full potential of such architectures. We foresee two particular challenges that need to be addressed in the near future, the description of the interfaces to interact with the state and the definition of the requirements of the databases storing the decoupled-state.

7.2.1 State Interfaces

SDN defines clear interfaces (e.g. northbound, southbound) between the different SDN components [48, 61, 132]. In common SDN architectures there is no well-defined interface between state and control since there is no clear boundary between them. There is no need to explicitly define a control⇔state interface since they are considered parts of a single entity. In this context, each SDN approach defines the interaction of control and state within the controller according to their own particular needs. However, logically dissociating the state from the controller introduces the need to define a clear interface between them. This is comparable to the decoupling of the control-plane from the data-plane in early days of SDN that defined the southbound interface. Similarly, it is also closely related to the extraction of the control applications from the controller that introduced the northbound interface in the SDN paradigm.

Furthermore, once the state is logically outside the controller it is possible to define additional interfaces with other SDN components. It is specially interesting to consider the direct interaction of the state with the data-plane devices. In this sense, this thesis investigated the interaction of the controller with a disjoint state entity in Chapter 4 and the direct state exchange between the state database and the data-plane devices in Chapter 3. Despite this initial research, further investigation is required to define the requirements of the control⇔state and state⇔data interfaces.

On one hand, in many use-cases the interface to exchange state between the controller and the state database can leverage on the common database interfaces available today. This is the case for instance in our proposal in Section 3.5.2 where the controller connects to a Cassandra-based state database making use of a regular Cassandra client driver. However, the requirements of this interface needs to be carefully analyzed per use-case and scenario. On the other hand, the interface to expose the state to the data-plane devices likely needs to be fast and lightweight, as discussed in Chapter 3. This also applies to the controller⇔state interface when the controller requires of fast and frequent queries to a remote decoupled-state. This is the case of the decentralized controlled introduced in Chapter 4.

This thesis proposes to use LISP as the state exchange interface for the aforementioned cases where a regular database driver is not the most suitable approach. We choose LISP to fulfill this role due to its lightweight approach and the easiness to leverage on its already defined architecture and signaling. However, we believe that further approaches should be considered in the future since different cases may require different approximations. Some examples of other candidates that, depending on the scenario, may be taken into account are gRPC [46] + ProtoBuff [117] and NETCONF [24] + YANG [6]. The former presents an interesting short-lived connection-oriented approach. It can help to mitigate the inherent drawbacks of connectionless proposals (e.g. LISP) without adding the common overhead of connection-based protocols. The latter offers a full connection-oriented proposal, with the overhead that it implies. However, its wide adoption allows it to be used on scenarios where no other approach is feasible. Furthermore, the flexible YANG datamodels could be directly mapped to the state structure on the database which opens interesting possibilities. For instance, to automatically generate model-based interfaces on demand in the manner of OpenDaylight [89].

7.2.2 State Databases

This work presents network architectures that heavily depend on databases and we believe that further research and analysis is needed to better understand the implications of this dependence. It should be noted that a partial database dependency has been in place through the different iterations of SDN controllers [47, 71, 59, 5]. As an example, the authors of ONOS [5] provide a large discussion on the state tradeoffs addressed by their architecture and describe the different database choices they considered in order fit the requirements of their scenario.

However, the logical decoupling of state from control and the resulting extraction of the state database from the logical boundaries of the controller stresses the role of the state database. Taking the state out alleviates the scalability requirements on the controller, but moves them to the state database. Therefore, the scalability of the SDN architecture is no longer mostly tied to the scalability of the controller, but rather to the scalability of the state database. Similarly, some technical requirements and constrains previously associated to the SDN controller are now moved to the state database.

In this sense, existing research on SDN controllers has to be revisited from a decoupled-state point of view, to asses whether the results of that research remain valid for decoupled-state architectures. Particularly interesting works are those that focus on aspects directly related to the state, for instance the implications of a distributed state across different partitions [76], the topological placement of these partitions [53], how to distribute the specific state across them [112] or how to ensure safe state transactions on a shared environment [10]. As an example, this thesis has already shown in section 3.7 how the topological location of the state has an important impact on the performance of decoupled-state SDN architectures.

Furthermore, decoupled-state architectures require making an efficient usage of the underlying database technology (as briefly discussed in Section 3.6). First, per each use-case and scenario the architecture has to find a database that suits its needs in terms of the CAP theorem (Consistency, Availability and Partition tolerance) [8, 7]. In common SDN deployments this is usually handled via solutions that offer high availability and partition tolerance by means of only offering eventual consistency. This is the case for the state database (e.g. Cassandra) that we use in Chapter 3 and Chapter 4. Second, the database needs to meet the architecture requirements in terms of available features, for instance how it handles merging overlapping information or if it allows to define callbacks functions to be triggered when certain state is modified. As an example, in Section 3.5.3 we rely on Cassandra triggers [136] to notify state updates to data-plane nodes.

Finally, the database deployment must fulfill the performance requirements of the architecture in both throughput (request per second) and delay (time per request). Decoupled-state architectures potentially introduce an additional burden on the latter since in some scenarios the state can be not only decoupled but also topologically far from the controller or data-plane devices. This could add extra latency to the state retrieval or modification and should be carefully considered during the design of decoupled-state architectures.

7.2.3 Long Term Research

The decoupling of state from control enables new research possibilities to be considered in the long term. Particularly, at the time of this writing we are starting to explore the application of Machine Learning (ML) techniques to extract knowledge from the decoupled-state. We believe that the vision described by D. Clark et al. in the paper *A Knowledge Plane for the Internet* [15] can be accomplished today by leveraging on the SDN architectures. Within this context, decoupled-state SDN architectures may play a major role.

In general, the state centralization brought by SDN makes possible to use ML-based solutions that were not feasible to apply over a distributed setup. Furthermore, the decoupling and isolation of the state proposed by decoupled-state architectures eases its parsing by ML applications. In a decoupled-state architecture the state is directly exposed to third parties, thus a ML application can extract knowledge from the state database while minimizing the disruption of the system. This generated knowledge can be later used to optimize different metrics across the system. Moreover, due to the open access to the database, it is easy for an application leveraging on the extracted knowledge to modify the network state directly. The field of applying ML for network control loops is still open research and the application of it to decoupled-state SDN architectures remains to be explored in the future.

Bibliography

[1] Amazon AWS (2016). Amazon Web Services. https://aws.amazon.com/. Accessed: 2016-04-28.

[2] Android (2016). Google Android. http://android.com/. Accessed: 2016-04-20.

[3] Apache (2004). Apache Software License 2.0. http://www.apache.org/licenses/LICENSE-2.0.

[4] Barkai, S., Farinacci, D., Meyer, D., Maino, F., Ermagan, V., Rodriguez-Natal, A., and Cabellos-Aparicio, A. (2015). LISP Based FlowMapping for Scaling NFV. Internet-Draft draft-barkai-lisp-nfv-07, Internet Engineering Task Force. Work in Progress.

[5] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., et al. (2014). ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM.

[6] Bjorklund, M. (2010). YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, RFC Editor. http://www.rfc-editor.org/rfc/rfc6020.txt.

[7] Brewer, E. (2012). CAP twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29.

[8] Brewer, E. A. (2000). Towards Robust Distributed Systems (Abstract). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 7–, New York, NY, USA. ACM.

[9] Callon, R. (1990). Use of OSI IS-IS for routing in TCP/IP and dual environments. RFC 1195, RFC Editor. http://www.rfc-editor.org/rfc/rfc1195.txt.

[10] Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. (2013). Software transactional networking: Concurrent and consistent policy composition. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 1–6. ACM.

[11] Casado, M., Foster, N., and Guha, A. (2014). Abstractions for software-defined networks. *Communications of the ACM*, 57(10):86–95.

[12] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM.

[13] Casado, M., Koponen, T., Shenker, S., and Tootoonchian, A. (2012). Fabric: a retrospective on evolving SDN. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 85–90. ACM.

[14] Cerrato, I., Palesandro, A., Risso, F., Suñé, M., Vercellone, V., and Woesner, H. (2015). Toward dynamic virtualized network services in telecom operator networks. *Computer Networks*, 92:380–395.

[15] Clark, D. D., Partridge, C., Ramming, J. C., and Wroclawski, J. T. (2003). A knowledge plane for the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10. ACM.

[16] Coras, F., Cabellos-Aparicio, A., and Domingo-Pascual, J. (2012). An analytical model for the LISP cache size. In *NETWORKING 2012*, pages 409–420. Springer.

[17] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

[18] Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. Technical report, DTIC Document.

[19] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. (2013). Towards an elastic distributed SDN controller. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 7–12. ACM.

[20] Doria, A., Salim, J. H., Haas, R., Khosravi, H., Wang, W., Dong, L., Gopal, R., and Halpern, J. (2010). Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810, RFC Editor. http://www.rfc-editor.org/rfc/rfc5810.txt.

[21] Droms, R. (1997). Dynamic Host Configuration Protocol. RFC 2131, RFC Editor. http://www.rfc-editor.org/rfc/rfc2131.txt.

[22] Eddy, W. M. (2004). At what layer does mobility belong? *Communications Magazine, IEEE*, 42(10):155–159.

[23] Emmerich, P., Raumer, D., Wohlfart, F., and Carle, G. (2014). Performance characteristics of virtual switching. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 120–125. IEEE.

[24] Enns, R., Bjorklund, M., Schoenwaelder, J., and Bierman, A. (2011). Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard).

[25] Ermagan, V., Farinacci, D., Lewis, D., Skriver, J., Maino, F., and White, C. (2016). NAT traversal for LISP. Internet-Draft draft-ermagan-lisp-nat-traversal-10, Internet Engineering Task Force. Work in Progress.

[26] Ermagan, V., Rodriguez-Natal, A., Coras, F., Moberg, C., Cabellos-Aparicio, A., and Maino, F. (2015). LISP Configuration YANG Model. Internet-Draft draft-ietf-lisp-yang-01, Internet Engineering Task Force. Work in Progress.

[27] ETSI Group Specification (2014a). Network Functions Virtualisation (NFV); Architectural Framework. Specification V1.2.1, European Telecommunications Standards Institute (ETSI). http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[28] ETSI Group Specification (2014b). Network Functions Virtualisation (NFV); Management and Orchestration . Specification V1.1.1, European Telecommunications Standards Institute (ETSI). http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.

[29] Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., and Estrin, D. (2010). Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM.

[30] Farinacci, D., Fuller, V., Meyer, D., and Lewis, D. (2013). The Locator/ID Separation Protocol (LISP). RFC 6830, RFC Editor. http://www.rfc-editor.org/rfc/rfc6830.txt.

[31] Farinacci, D., Kowal, M., and Lahiri, P. (2016a). LISP Traffic Engineering Use-Cases. Internet-Draft draft-farinacci-lisp-te-10, Internet Engineering Task Force. Work in Progress.

[32] Farinacci, D., Lewis, D., Meyer, D., and White, C. (2016b). LISP Mobile Node. Internet-Draft draft-meyer-lisp-mn-14, Internet Engineering Task Force. Work in Progress.

[33] Farinacci, D., Meyer, D., and Snijders, J. (2016c). LISP Canonical Address Format (LCAF). Internet-Draft draft-ietf-lisp-lcaf-13, Internet Engineering Task Force. Work in Progress.

[34] Farrel, A., Vasseur, J.-P., and Ash, J. (2006). A Path Computation Element (PCE)-Based Architecture. RFC 4655, RFC Editor. http://www.rfc-editor.org/rfc/rfc4655.txt.

[35] Feamster, N., Rexford, J., and Zegura, E. (2014). The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98.

[36] Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M. J., Katta, N. P., Monsanto, C., Reich, J., Rexford, J., Schlesinger, C., et al. (2013). Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134.

[37] Fuller, V. and Farinacci, D. (2013). Locator/ID Separation Protocol (LISP) Map-Server Interface. RFC 6833, RFC Editor.

[38] Fuller, V., Lewis, D., Ermagan, V., Jain, A., and Smirnov, A. (2016). LISP Delegated Database Tree. Internet-Draft draft-ietf-lisp-ddt-06, Internet Engineering Task Force. Work in Progress.

[39] Galvani, A., Rodriguez-Natal, A., Cabellos-Aparicio, A., and Risso, F. (2014). LISP-ROAM: network-based host mobility with LISP. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 19–24. ACM.

[40] Garg, P. and Wang, Y. (2015). NVGRE: Network Virtualization Using Generic Routing Encapsulation. RFC 7637, RFC Editor.

[41] GBP (2015). Group Based Policy abstractions for OpenStack. https://wiki.openstack.org/GroupBasedPolicy. Accessed: 2015-03-13.

[42] Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S., and Akella, A. (2015). OpenNF: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 44(4):163–174.

[43] Goldschlag, D. M., Reed, M. G., and Syverson, P. F. (1996). Hiding routing information. In *Information Hiding*, pages 137–150. Springer.

[44] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54.

[45] Greene, K. (2009). TR10: Software-defined networking. *Technology Review (MIT)*. http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/.

[46] gRPC (2016). Google gRPC. http://www.grpc.io/. Accessed: 2016-04-19.

[47] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110.

[48] Haleplidis, E., Pentikousis, K., Denazis, S., Salim, J. H., Meyer, D., and Koufopavlou, O. (2015). Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, RFC Editor. http://www.rfc-editor.org/rfc/rfc7426.txt.

[49] Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE*, 53(2):90–97.

[50] Han, S., Liu, V., Pu, Q., Peter, S., Anderson, T., Krishnamurthy, A., and Wetherall, D. (2013). Expressive privacy control with pseudonyms. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 291–302. ACM.

[51] Hawilo, H., Shami, A., Mirahmadi, M., and Asal, R. (2014). NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *Network, IEEE*, 28(6):18–26.

[52] Heller, B. et al. (2009). OpenFlow Switch Specification. Version 1.0. 0 (Wire Protocol 0x01).

[53] Heller, B., Sherwood, R., and McKeown, N. (2012). The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM.

[54] Hinrichs, T. L., Gude, N. S., Casado, M., Mitchell, J. C., and Shenker, S. (2009). Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 1–10. ACM.

[55] Hoefling, M., Menth, M., and Hartmann, M. (2013). A survey of mapping systems for locator/identifier split internet routing. *Communications Surveys & Tutorials, IEEE*, 15(4):1842–1858.

[56] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Annual Technical Conference*, volume 8, page 9.

[57] Iannone, L., Saucez, D., and Bonaventure, O. (2013). Locator/ID Separation Protocol (LISP) Map-Versioning. RFC 6834, RFC Editor.

[58] Intel DPDK (2016). Intel Data Plane Development Kit. http://openvpn.net. Accessed: 2016-04-28.

[59] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al. (2013). B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM.

[60] Jakab, L., Cabellos-Aparicio, A., Coras, F., Saucez, D., and Bonaventure, O. (2010). LISP-TREE: a DNS hierarchy to support the lisp mapping system. *Selected Areas in Communications, IEEE Journal on*, 28(8):1332–1343.

[61] Jarschel, M., Zinner, T., Hoßfeld, T., Tran-Gia, P., and Kellerer, W. (2014). Interfaces, attributes, and use cases: A compass for SDN. *Communications Magazine, IEEE*, 52(6):210–217.

[62] JNI (2016). Java Native Interface. http://docs.oracle.com/javase/7/docs/technotes/guides/jni/. Accessed: 2016-04-20.

[63] Johnson, D., Perkins, C., and Arkko, J. (2004). Mobility Support in IPv6. RFC 3775, RFC Editor. http://www.rfc-editor.org/rfc/rfc3775.txt.

[64] Kachris, C. and Tomkos, I. (2012). A survey on optical interconnects for data centers. *Communications Surveys & Tutorials, IEEE*, 14(4):1021–1036.

[65] Kernighan, B. W., Ritchie, D. M., and Ejeklint, P. (1988). *The C programming language*, volume 2. prentice-Hall Englewood Cliffs.

[66] Kim, H. and Feamster, N. (2013). Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119.

[67] Klein, D., Hartmann, M., and Menth, M. (2010). NAT traversal for LISP mobile node. In *Proceedings of the Re-Architecting the Internet Workshop*, page 8. ACM.

[68] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. (2000). The Click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297.

[69] Koodli, R. (2005). Fast Handovers for Mobile IPv6. RFC 4068, RFC Editor.

[70] Koodli, R. (2007). IP Address Location Privacy and Mobile IPv6: Problem Statement. RFC 4882, RFC Editor.

[71] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. (2010). Onix: A Distributed Control Platform for Large-scale Production Networks. In *OSDI*, volume 10, pages 1–6.

[72] Kreeger, L. and Elzur, U. (2016). Generic Protocol Extension for VXLAN. Internet-Draft draft-ietf-nvo3-vxlan-gpe-02, Internet Engineering Task Force. Work in Progress.

[73] Krejci, R. (2013). Building NETCONF-enabled network management systems with libnetconf. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 756–759. IEEE.

[74] Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

[75] Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

[76] Levin, D., Wundsam, A., Heller, B., Handigol, N., and Feldmann, A. (2012). Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 1–6. ACM.

[77] Lewis, D., Meyer, D., Farinacci, D., and Fuller, V. (2013). Interworking between Locator/ID Separation Protocol (LISP) and Non-LISP Sites. RFC 6832, RFC Editor.

[78] libnetconf (2016). C NETCONF library. https://github.com/CESNET/libnetconf. Accessed: 2016-04-22.

[79] Linux (2016). The Linux kernel archives. http://kernel.org/. Accessed: 2016-04-20.

[80] lisp4.net (2016). The LISP beta-network project. http://lisp4.net. Accessed: 2016-03-25.

[81] lisp.cisco.com (2016). Cisco LISP. http://lisp.cisco.com. Accessed: 2016-04-10.

[82] LISPmob (2016). LISPmob open-source project. http://www.openoverlayrouter.org/lispmob/. Accessed: 2016-04-20.

[83] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and Wright, C. (2014). Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348 (Informational).

[84] Maino, F., Ermagan, V., Cabellos-Aparicio, A., and Saucez, D. (2016). LISP-Security (LISP-SEC). Internet-Draft draft-ietf-lisp-sec-10, Internet Engineering Task Force. Work in Progress.

[85] Mathy, L. and Iannone, L. (2008). LISP-DHT: Towards a DHT to map identifiers onto locators. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 61. ACM.

[86] Matias, J., Garay, J., Toledo, N., Unzilla, J., and Jacob, E. (2015). Toward an SDN-enabled NFV architecture. *Communications Magazine, IEEE*, 53(4):187–193.

[87] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

[88] McQuillan, J. M., Richer, I., and Rosen, E. C. (1980). The new routing algorithm for the ARPANET. *Communications, IEEE Transactions on*, 28(5):711–719.

[89] Medved, J., Varga, R., Tkacik, A., and Gray, K. (2014). Opendaylight: Towards a model-driven sdn controller architecture. In *2014 IEEE 15th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–6. IEEE.

[90] Meneses, F., Corujo, D., Guimaraes, C., and Aguiar, R. L. (2015). Extending SDN to End Nodes Towards Heterogeneous Wireless Mobility. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE.

[91] Menth, M., Klein, D., and Hartmann, M. (2010). Improvements to LISP mobile node. In *Teletraffic Congress (ITC), 2010 22nd International*, pages 1–8. IEEE.

[92] Metcalfe, R. M. and Boggs, D. R. (1976). Ethernet: distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404.

[93] Monsanto, C., Reich, J., Foster, N., Rexford, J., and Walker, D. (2013). Composing software defined networks. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 1–13.

[94] Moore, N. (2006). Optimistic Duplicate Address Detection (DAD) for IPv6. RFC 4429, RFC Editor.

[95] Moy, J. (1998). OSPF Version 2. STD 54, RFC Editor. http://www.rfc-editor.org/rfc/rfc2328.txt.

[96] Narten, T., Draves, R., and Krishnan, S. (2007). Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941, RFC Editor.

[97] Nordmark, E., Chakrabarti, S., and Laganier, J. (2007). IPv6 Socket API for Source Address Selection. RFC 5014, RFC Editor.

[98] Nordström, E., Shue, D., Gopalan, P., Kiefer, R., Arye, M., Ko, S. Y., Rexford, J., and Freedman, M. J. (2012). Serval: An end-host stack for service-centric networking. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 7–7. USENIX Association.

[99] ODL-LISP (2015). OpenDaylight LISP Flow Mapping Project. https://wiki.opendaylight.org/view/OpenDaylight_Lisp_Flow_Mapping:Main. Accessed: 2015-03-13.

[100] Open Overlay Router (2016). Open Overlay Router open-source project. http://openoverlayrouter.org/. Accessed: 2016-03-25.

[101] Open Source MANO (OSM) (2016). Open Source NFV Management and Orchestration (MANO). https://osm.etsi.org/. Accessed: 2016-04-26.

[102] Open VPN (2016). OpenVPN - Open Source VPN. http://openvpn.net. Accessed: 2016-04-28.

[103] OpenDaylight (2015). OpenDaylight Project. http://www.opendaylight.org/. Accessed: 2015-03-13.

[104] OpenLISP (2016). OpenLISP project. http://www.openlisp.org/. Accessed: 2016-03-25.

[105] OpenMANO (2016). Telefonica OpenMANO open-source project. https://github.com/nfvlabs/openmano/. Accessed: 2016-03-25.

[106] OpenStack (2015). OpenStack Open Source Cloud Computing Software. https://www.openstack.org/. Accessed: 2015-03-13.

[107] OpenWrt (2016). OpenWrt: a Linux distribution for embedded devices. http://openwrt.org/. Accessed: 2016-04-20.

[108] Paul, S. and Jain, R. (2012). OpenADN: Mobile apps on global clouds using openflow and software defined networking. In *Globecom Workshops (GC Wkshps), 2012 IEEE*, pages 719–723. IEEE.

[109] Perkins, C. (2002). IP Mobility Support for IPv4. RFC 3344, RFC Editor. http://www.rfc-editor.org/rfc/rfc3344.txt.

[110] Perlman, R. (1985). An algorithm for distributed computation of a spanningtree in an extended lan. In *ACM SIGCOMM Computer Communication Review*, volume 15, pages 44–53. ACM.

[111] Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., et al. (2015). The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130.

[112] Phemius, K., Bouet, M., and Leguay, J. (2014). Disco: Distributed multi-domain sdn controllers. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4. IEEE.

[113] Phung, D., Secci, S., Saucez, D., and Iannone, L. (2014). The OpenLISP control plane architecture. *Network, IEEE*, 28(2):34–40.

[114] Pongrácz, G., Molnar, L., and Kis, Z. L. (2013). Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 62–67. IEEE.

[115] Postel, J. (1981a). Internet Protocol. STD 5, RFC Editor. http://www.rfc-editor.org/rfc/rfc791.txt.

[116] Postel, J. (1981b). Transmission Control Protocol. STD 7, RFC Editor. http://www.rfc-editor.org/rfc/rfc793.txt.

[117] ProtoBuff (2016). Google Protocol Buffers. https://developers.google.com/protocol-buffers/. Accessed: 2016-04-19.

[118] Qiu, Y., Zhao, F., and Koodli, R. (2010). Mobile IPv6 Location Privacy Solutions. RFC 5726, RFC Editor.

[119] Rabl, T., Gómez-Villamor, S., Sadoghi, M., Muntés-Mulero, V., Jacobsen, H.-A., and Mankovskii, S. (2012). Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment*, 5(12):1724–1735.

[120] Rekhter, Y., Li, T., and Hares, S. (2006). A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor. http://www.rfc-editor.org/rfc/rfc4271.txt.

[121] Risso, F. and Cerrato, I. (2012). Customizing data-plane processing in edge routers. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 114–120. IEEE.

[122] Rodriguez-Natal, A., Cabellos-Aparicio, A., Barkai, S., Ermagan, V., Lewis, D., Maino, F., and Farinacci, D. (2016a). LISP support for Multi-Tuple EIDs. Internet-Draft draft-rodrigueznatal-lisp-multi-tuple-eids-01, Internet Engineering Task Force. Work in Progress.

[123] Rodriguez-Natal, A., Cabellos-Aparicio, A., Ermagan, V., Maino, F., and Barkai, S. (2016b). MS-originated SMRs. Internet-Draft draft-rodrigueznatal-lisp-ms-smr-01, Internet Engineering Task Force. Work in Progress.

[124] Rodriguez-Natal, A., Jakab, L., Ermagan, V., Natarajan, P., Maino, F., and Cabellos-Aparicio, A. (2015a). Location and identity privacy for LISP-MN. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5260–5265. IEEE.

[125] Rodriguez-Natal, A., Jakab, L., Portoles, M., Ermagan, V., Natarajan, P., Maino, F., Meyer, D., and Cabellos-Aparicio, A. (2013). LISP-MN: mobile networking through LISP. *Wireless personal communications*, 70(1):253–266.

[126] Rodriguez-Natal, A., Portoles-Comeras, M., Ermagan, V., Lewis, D., Farinacci, D., Maino, F., and Cabellos-Aparicio, A. (2015b). LISP: a southbound SDN protocol? *Communications Magazine, IEEE*, 53(7):201–207.

[127] Rosen, E., Viswanathan, A., and Callon, R. (2001). Multiprotocol Label Switching Architecture. RFC 3031, RFC Editor. http://www.rfc-editor.org/rfc/rfc3031.txt.

[128] Route Views (2016). University of Oregon Route Views Project. http://www.routeviews.org/. Accessed: 2016-04-29.

[129] Saucez, D., Iannone, L., Bonaventure, O., and Farinacci, D. (2012). Designing a deployable internet: The locator/identifier separation protocol. *Internet Computing, IEEE*, 16(6):14–21.

[130] Schmid, S. and Suomela, J. (2013). Exploiting locality in distributed SDN control. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 121–126. ACM.

[131] Sezer, S., Scott-Hayward, S., Chouhan, P.-K., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., and Rao, N. (2013). Are we ready for SDN? Implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43.

[132] Shin, M.-K., Nam, K.-H., and Kim, H.-J. (2012). Software-defined networking (SDN): A reference architecture and open APIs. In *ICT Convergence (ICTC), 2012 International Conference on*, pages 360–361. IEEE.

[133] Shoch, J. (1978). Inter-network naming, addressing, and routing. In *COMPCON, IEEE Computer Society, Fall*, volume 5.

[134] Takahashi, H. and Minohara, T. (2012). Enhancing location privacy in Mobile IPv6 by using redundant home agents. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 451–454. IEEE.

[135] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., and Minden, G. J. (1997). A survey of active network research. *Communications Magazine, IEEE*, 35(1):80–86.

[136] Triggers (2016). Cassandra triggers. http://www.datastax.com/dev/blog/whats-new-in-cassandra-2-0-prototype-triggers-support. Accessed: 2016-04-20.

[137] Tsirtsis, G., Soliman, H., Montavont, N., Giaretta, G., and Kuladinithi, K. (2011). Flow Bindings in Mobile IPv6 and Network Mobility (NEMO) Basic Support. RFC 6089, RFC Editor.

[138] TUN/TAP (2016). Linux Kernel documentation. TUN/TAP device driver. http://www.kernel.org/doc/Documentation/networking/tuntap.txt. Accessed: 2016-04-29.

[139] Vasseur, J. and Roux, J. L. (2009). Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440, RFC Editor. http://www.rfc-editor.org/rfc/rfc5440.txt.

[140] Vissicchio, S., Vanbever, L., and Bonaventure, O. (2014). Opportunities and research challenges of hybrid software defined networks. *ACM SIGCOMM Computer Communication Review*, 44(2):70–75.

[141] VPN-API (2016). Google Android VPN Service Development Reference. http://developer.android.com/reference/android/net/VpnService.html. Accessed: 2016-04-28.

[142] Weber, I., Zagheni, E., et al. (2013). Studying inter-national mobility through IP geolocation. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 265–274. ACM.

[143] Yap, K.-K., Huang, T.-Y., Kobayashi, M., Yiakoumis, Y., McKeown, N., Katti, S., and Parulkar, G. (2012). Making use of all the networks around us: a case study in android. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, pages 19–24. ACM.

[144] Yeganeh, S. H., Tootoonchian, A., and Ganjali, Y. (2013). On scalability of software-defined networking. *Communications magazine, IEEE*, 51(2):136–141.

[145] Zhao, Y., Iannone, L., and Riguidel, M. (2014). Software switch performance factors in network virtualization environment. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 468–470. IEEE.

Appendix A

Complete List of Publications

A.1 Related Publications

A.1.1 Journals

- Rodriguez-Natal, Alberto, Lorand Jakab, Marc Portoles-Comeras, Vina Ermagan, Preethi Natarajan, Fabio Maino, and Albert Cabellos-Aparicio. "LISP-MN: mobile networking through LISP." *Wireless personal communications* 70.1 (2013): 253-266.

- Rodriguez-Natal, Alberto, Marc Portoles-Comeras, Vina Ermagan, Darrel Lewis, Dino Farinacci, Fabio Maino, and Albert Cabellos-Aparicio. "LISP: a southbound SDN protocol?" *Communications Magazine, IEEE* 53.7 (2015): 201-207.

Under Review

- Rodriguez-Natal, Alberto, Vina Ermagan, Kien Nguyen, Sharon Barkai, Yusheng Ji, Fabio Maino, and Albert Cabellos-Aparicio. “SDN for End-Nodes.”

- Rodriguez-Natal, Alberto, Vina Ermagan, Ariel Noy, Ajay Sahai, Gidi Kaempfer, Sharon Barkai, Fabio Maino, and Albert Cabellos-Aparicio. “Global state, local decisions: Decentralized NFV for ISPs via enhanced SDN.”

- Rodriguez-Natal, Alberto, Florin Coras, Albert Lopez-Bresco, Lorand Jakab, Marc Portoles-Comeras, Preethi Natarajan, Vina Ermagan, David Meyer, Dino Farinacci, Fabio Maino, and Albert Cabellos-Aparicio. “OpenOverlayRouter: Architecture and Performance.”

A.1.2 Conferences

- Rodriguez-Natal, Alberto, Lorand Jakab, Vina Ermagan, Preethi Natarajan, Fabio Maino, and Albert Cabellos-Aparicio. "Location and identity privacy for LISP-MN." *International Conference on Communications (ICC), IEEE,* 2015.

A.1.3 Internet Drafts

- Ermagan, Vina, Alberto Rodriguez-Natal, Florin Coras, Albert Cabellos-Aparicio, and Fabio Maino. "LISP Configuration YANG Model", *draft-ietf-lisp-yang-01*, December 2015, (work in progress).
  https://tools.ietf.org/html/draft-ietf-lisp-yang-01

- Rodriguez-Natal, Alberto, Albert Cabellos-Aparicio, Sharon Barkai, Vina Ermagan, Darrel Lewis, Fabio Maino, and Dino Farinacci. "LISP support for Multi-Tuple EIDs", *draft-rodrigueznatal-lisp-multi-tuple-eids-01*, January 2016, (work in progress).
  https://tools.ietf.org/html/draft-rodrigueznatal-lisp-multi-tuple-eids-01

- Barkai, Sharon, Dino Farinacci, David Meyer, Fabio Maino, Vina Ermagan, Alberto Rodriguez-Natal, and Albert Cabellos-Aparicio. "LISP Based FlowMapping for Scaling NFV", *draft-barkai-lisp-nfv-07*, December 2015, (work in progress).
  https://tools.ietf.org/html/draft-barkai-lisp-nfv-07

- Rodriguez-Natal, Alberto, Albert Cabellos-Aparicio, Vina Ermagan, Fabio Maino, and Sharon Barkai. "MS-originated SMRs", *draft-rodrigueznatal-lisp-ms-smr-01*, April 2016, (work in progress).
  https://tools.ietf.org/html/draft-rodrigueznatal-lisp-ms-smr-01

A.1.4 Talks

- “LISPflow: an SDN enabler”, *IETF 87 SDN RG*, Berlin, Germany, July 2013.
  https://datatracker.ietf.org/meeting/87/agenda/sdnrg

- “LISP & LISPmob: Overview and Use Cases” [1][2], *RIPE 69 Routing WG*, London, UK, November 2014.
  https://ripe69.ripe.net/programme/meeting-plan/routing-wg/

[1]invited by the RIPE Academic Initiative Collaboration (RACI) program.
[2]https://ripe69.ripe.net/programme/raci/

- “LISP YANG model”, *IETF 92 LISP WG*, Dallas, TX, USA, March 2015.
  https://datatracker.ietf.org/meeting/92/agenda/lisp/
- “An Over-The-Top SDN Architecture for Mobile Nodes and Home Routers”, *IETF 92 SDN RG*, Dallas, TX, USA, March 2015.
  https://datatracker.ietf.org/meeting/92/agenda/sdnrg
- “LISP subscription”, *IETF 94 LISP WG*, Yokohama, Japan, November 2015.
  https://datatracker.ietf.org/meeting/94/agenda/lisp/
- “Open Overlay Lab: SDN via programmable overlays”, *European Commission Info Day on 5G-PPP Phase 2*, Bologna, Italy, March 2016.
  https://5g-ppp.eu/5g-ppp-phase-2-information-day-and-stakeholders-event/

A.1.5 Code

- Core developer and maintainer at the OpenOverlayRouter project (formerly LISPmob), an open-source implementation to deploy programmable overlays.
  http://www.openoverlayrouter.org/
  https://github.com/OpenOverlayRouter
- Contributor of the original LISP northbound interface for the OpenDaylight SDN controller.
  https://github.com/opendaylight/lispflowmapping

A.2 Other Publications

A.2.1 Workshops

- Galvani, Andrea, Alberto Rodriguez-Natal, Albert Cabellos-Aparicio, and Fulvio Risso, "LISP-ROAM: network-based host mobility with LISP." *Proc. of the 9th ACM workshop on Mobility in the evolving internet architecture. ACM*, 2014.

A.2.2 Book Chapters

- Papadimitriou, Dimitri, Florin Coras, Alberto Rodriguez, Valentin Carela, Davide Careglio, Lluís Fàbrega, Pere Vilà, and Piet Demeester. "Iterative research method applied to the design and evaluation of a dynamic multicast routing scheme." *In Measurement Methodology and Tools*, pp. 107-126. Springer Berlin Heidelberg, 2013.

A.2.3 Patents

- Portoles-Comeras, Marc, Preethi Natarajan, Alberto Rodriguez-Natal, Fabio Maino, Albert Cabellos-Aparicio, Vasileios Lakafosis, and Lorand Jakab. “Multipath Provisioning of L4-L7 Traffic in a Network.” Patent Application. US 14/612,691 (03-Feb-2015). EP 14465524.8 (27-Oct-2014).

A.2.4 Internet Drafts

- Rodriguez-Natal, Alberto, Albert Cabellos-Aparicio, Marc Portoles-Comeras, Michael Kowal, Darrel Lewis, and Fabio Maino. "LISP-OAM: Use cases and requirements", *draft-rodrigueznatal-lisp-oam-03*, December 2015, (work in progress).
https://tools.ietf.org/html/draft-rodrigueznatal-lisp-oam-03

- Cabellos, Albert, Sharon Barkai, Barak Perlman, Vina Ermagan, Fabio Maino, and Alberto Rodriguez-Natal. "Map-Assisted SFC Proxy using LISP", *draft-cabellos-sfc-map-assisted-proxy-00*, October 2015, (work in progress).
https://tools.ietf.org/html/draft-cabellos-sfc-map-assisted-proxy-00