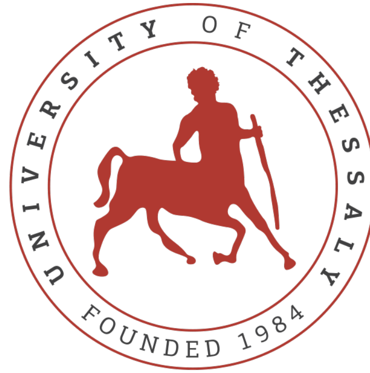


UNIVERSITY OF THESSALY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
MSc STUDIES IN SCIENCE AND TECHNOLOGY OF ECE

Master Thesis

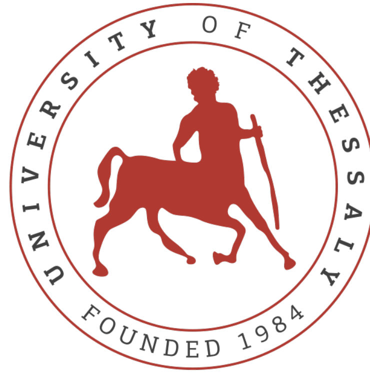
Design and Implementation of Networking Protocols with the P4 Programming Language and the ONOS SDN Controller

Panagiotis Karamichailidis

supervised by
Athanasios KORAKIS,
Associate Professor

Volos, 2021

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Τμημα Ηλεκτρολογων Μηχανικων & Μηχανικων Υπολογιστων
Προγραμμα Μεταπτυχιακων Σπουδων στην Επιστημη & Τεχνολογια
Ηλεκτρολογου Μηχανικου & Μηχανικου Υπολογιστων

Μεταπτυχιακή Διπλωματική Εργασία

Σχεδιασμός και Υλοποίηση δικτυακών πρωτοκόλλων με χρήση της γλώσσας προγραμματισμού P4 και του ONOS διαχειριστή δικτύων προγραμματιζόμενων από λογισμικό

Παναγιώτης Καραμιχαηλίδης

επιβλέπων
Αθανάσιος Κορακης,
Αναπληρωτής Καθηγητής

Βόλος, 2021

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής μου εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Αθανάσιο Κοράκη για την ευκαιρία που μου έδωσε αναθέτοντας μου το συγκεκριμένο θέμα, καθώς, επίσης, και για την εμπιστοσύνη που μου έδειξε.

Είμαι ευγνώμων στον κ. Νικόλαο Μακρή, η πολύτιμη βοήθειά του και οι γνώσεις που απέκτησα χάρη σε αυτόν, καθ' όλη τη διάρκεια μιας άψογης συνεργασίας, σε πολυάριθμες ώρες διδακτικών συζητήσεων, είναι πολύτιμες. Η επιστημονική και ηθική ενθάρρυνση, αλλά και η συνεισφορά του στην εκπόνηση της παρούσας μελέτης είναι ανεκτίμητη.

Ακόμα, θα ήθελα να ευχαριστήσω όλους τους καθηγητές και συμφοιτητές ανεξαιρέτως για όλα αυτά που μου πρόσφεραν, ο καθένας με το δικό του τρόπο, στα χρόνια φοίτησής μου.

Θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου για την κατανόηση και την ενθάρρυνση που έδειξαν όλο αυτό το διάστημα, αλλά και την οικονομική υποστήριξη που μου παρείχαν όλα αυτά τα χρόνια.

Στην οικογένεια και τους φίλους μου.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

Περίληψη Το σημερινό Διαδίκτυο απαιτεί εύκολα διαχειρίσιμα και απλά επεκτάσιμα συστήματα ελέγχου δικτύου. Το Software-Defined Networking (SDN) είναι μια αναδυόμενη αρχιτεκτονική που στοχεύει στη δημιουργία ενός συστήματος για τις επερχόμενες ανάγκες, προσφέροντας έναν άμεσα προγραμματιζόμενο, ευέλικτο, κεντρικά διαχειριζόμενο και προγραμματισμένο τρόπο για τους χειριστές να ελέγχουν το δίκτυό τους [6]. Το SDN αποσυνδέει τις λειτουργίες ελέγχου δικτύου και προώθησης, γεγονός που διευκολύνει τη δημιουργία νέων αφαιρέσεων στη δικτύωση, απλοποιώντας τη διαχείριση και διευκολύνοντας την πρόοδο του δικτύου.

Οι συσκευές SDN μπορούν να προγραμματιστούν μέσω μιας ειδικής διεπαφής, με ένα συγκεκριμένο πρωτόκολλο, το οποίο είναι το OpenFlow [1]. Το μεγαλύτερο πρόβλημα με το OpenFlow είναι ότι δεν υποστηρίζει νέους ορισμούς κεφαλίδας (headers), κάτι που είναι απαραίτητο για τους χειριστές δικτύου να εφαρμόζουν νέες ενθυλακώσεις πακέτων (packet encapsulations). Για να ξεπεραστούν αυτά τα ζητήματα με το OpenFlow, δημιουργήθηκε μια νέα γλώσσα υψηλού επιπέδου: Programming Protocol-independent Packet Processor (P4). Αυτή η γλώσσα υποστηρίζει έναν πλήρως προγραμματιζόμενο πρόγραμμα ανάλυσης, ο οποίος μας καθιστά σε θέση να ορίσουμε νέες κεφαλίδες χωρίς πρόβλημα.

Σε αυτή τη διατριβή παρουσιάζουμε πώς τα P4 και SDN μπορούν να συνδυαστούν για να δημιουργήσουν μια καλή λύση για τους χειριστές δικτύου.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

Abstract In our day and age, Internet requires network control systems that are both easily manageable and extensible to cover contemporary needs [6]. To this end, a promising architecture, known as Software-Defined Networking (SDN), has been introduced, which offers several noteworthy advantages, such as directly programmability and flexibility for the operators to control their network. The creation of new abstractions in networking is easier, since SDN decouples the network control and forwarding functions. Furthermore, the management is simplified and the network advancement is easier.

However, there are many issues that arise using this technology, due to the fact that SDN devices are programmable through a dedicated interface using a particular protocol, known as the OpenFlow [1]. The biggest problem concerns the fact that OpenFlow does not support new header definitions, which is mandatory for network operators to apply new packet encapsulations. In order to address these issues, a new high-level language Programming Protocol-independent Packet Processors (P4) has been created. The main advantage of this technology is that it supports a fully programmable parser, which gives the opportunity to smoothly define new headers.

The objective in this thesis is the integration of P4 and SDN technologies in a combined framework in order to create a good solution for network operators.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Thesis Outline . . . 1

2 Software Defined Networking **2**
2.1 Overview of SDN . . . 2
2.2 SDN Architecture . . . 2
2.3 SDN Applications . . . 3
2.4 SDN Controllers . . . 4
2.5 Challenges Associated with SDN . . . 5

3 Network Functions Virtualization **6**
3.1 Overview of NFV . . . 6
3.2 VNFs . . . 6
3.3 Network Services in NFV . . . 6
3.4 The Benefits of NFV . . . 8
3.5 Challenges Associated with NFV . . . 8
3.6 NFV vs SDN . . . 8

4 Programming Protocol-Independent Packet Processors **9**
4.1 Overview . . . 9
4.2 Behavioral Model . . . 10
4.3 V1model . . . 11
4.3.1 Architecture . . . 11
4.3.2 Standard Metadata . . . 11
4.3.3 Program Template . . . 12
4.3.4 Basic Example . . . 13
4.3.5 Basic and Header Types . . . 14
4.3.6 Other Types . . . 15
4.3.7 Parser . . . 15
4.3.8 Simple Actions . . . 16
4.3.9 Tables . . . 17
4.3.10 Deparser . . . 19
4.4 P4 Runtime . . . 20

5 Open Network Operating System **22**
5.1 Overview of ONOS . . . 22
5.2 ONOS Design . . . 22
5.3 ONOS System Components . . . 23
5.4 Subsystem structure . . . 23
5.5 Network State Construction . . . 25
5.6 Device Subsystem . . . 26
5.7 Device Driver Subsystem . . . 27

6 Next Generation Software Defined Networking **28**
6.1 Stratum . . . 28
6.2 YANG . . . 37
6.3 OpenConfig . . . 39
6.3.1 gNMI . . . 40
6.4 ONOS as the control plane . . . 44
6.4.1 Enable Packet I/O . . . 45
6.4.2 L2 Bridging . . . 56
6.5 IPv6 Routing . . . 64
6.6 Segment Routing v6 . . . 81

7 Conclusion and Future Work **92**

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

1 Introduction

1.1 Motivation

Significant changes have been made to traditional networks, with the emergence of Software-Defined-Networking (SDN) [7]. SDN is a multiple structured architecture consisting of three layers, namely the infrastructure (or physical), the control and the application layer. Between the physical and the control layer, the most common protocol used for providing the interface, which is called Southbound APIs, is OpenFlow Protocol. Between the control and the application layer, the provided interface is called Northbound APIs. The controller, which represents the intelligent part or the "brain" of the network, configures and manages the flow of the network through network devices, which are present in the physical layer. Decoupling the forwarding hardware (data plane) from the control logic of the network (control plane), which is the basic idea of the approach, the integration of the network is achieved. Therefore, several limitations of traditional networks are addressed with this sophisticated technology, which offers promising tools for well-managed networks described by reliability, security and flexibility.

However, there are many issues to be considered about the use of SDN [12]. The main concern regards the required time for implementing new protocols or the extension of their functionality. The aforementioned tasks require a significant amount of time, as each protocol goes first through a long process concerning the IETF board. To define an official standard, an implementation in ASICs is needed, for which real world examples, such as VXLAN, suggest that this takes up to a couple of years to do so. On top of that, it is not certain whether or not protocol modifications will be needed. Another issue that needs to be taken into consideration when it comes to enterprise networks is their dependency upon network chip vendors. The emergence of P4 language is a good example of joined forces to overcome the above problems.

P4 (Programming Protocol-Independent Packet Processors) is a domain-specific language that is designed to fit on a great spectrum of applications, such as hardware/software switches, routers, network interface cards (NICs), or network appliances, offering a number of significant advantages [12], [33]. An important benefit of P4 language is that it gives the ability to implement a network stack in switching hardware using a basic set of tools. That means that user-defined keys can be associated with actions, counters and so on, operating abstractions as header types, parsers and tables. Furthermore, P4 language completely ignores the appearance of Ethernet or IP headers. It is the developer who informs silicon about the headers and how to parse, and to further match and construct a header for the outgoing packet and where to forward the packet.

Compared to other solutions, such as hardware solutions or traditional fixed-function switches, the aforementioned programmability of P4 language opens up completely new possibilities for enterprise, such as flexibility of network stack and upgradability that no longer demands the purchase of new switches since P4 software can be easily updated [12].

1.2 Thesis Outline

The rest of the thesis is organized as follows: Section 2 presents a briefly introduction to SDN; in section 3, NVF is presented, which is followed by a comparison of NVF to SDN; in section 4, P4 language is presented; section 5 presents ONOS an SDN controller; in section 6 some network protocols are implemented with the SDN and P4, whereas section 7 summarizes the conclusions of the current study, and provides plans for future work.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

2 Software Defined Networking

2.1 Overview of SDN

The success of particular technologies, as for example is cloud computing severely depends on network systems [2]. However, the slow development of scalable IT infrastructure, give rise to several issues, SDN technology comes to address such issues by giving new functions to the whole, network topology. Furthermore, administrators have the ability to abstract the underlying network infrastructure for applications and network services.

The commercialization of SDN, as well as its development and standardization, stems from The Open Networking Foundation (ONF), which is a non-profit consortium [3]. Based on ONF, the definition of SDN is the following: "SDN is an emerging network architecture where network control is decoupled from forwarding and is directly programmable.Per this definition, SDN is defined by two characteristics,namely decoupling of control and data planes, and programmability on the control plane".

In literature, a variety of benefits of SDN are mentioned [3]. For instance, a greater control of a network through programming is offered from the inherent decoupling of control plane from data plane. This, in turn, give, rise to other benefits of improved configuration and enhanced performance. Specifically, SDN control technology may bypass the barrier of layers by performing both a packet forwarding at a switching level and link tuning at a data link level. Also, SDN is capable of acquiring instantaneous network status, permitting a real-time centralized control of a network allowing for consideration of user defined policies, as well. The aforementioned benefit, give, the opportunity to further optimize network configurations and improve network performance, in general. SDN programmability also offers a handy platform for either experimenting new techniques or designing new ones.

SDN allows network programmability via the, API. The administrator has the freedom of configuring the devices dynamically as per the demands from the users. This brings agility and robustness to the functionality of the network [4] . The administrator has access to the centralized SDN controller to distribute policies to the switches that are connected to the controller. The dynamic updates of rules facilitate rerouting of traffic to improve network performance. Figure 1 illustrates the overview of network programmability in SDN network. It depicts the network administrator programming the SDN OpenFlow data plane devices via the SDN controller to schedule, reroute and to split, traffic across available links according to the user demands and the requirement in order to achieve optimal performance.

2.2 SDN Architecture

In the SDN architecture, the control and data forwarding functions are splitted and this splitting is referred to as dis-aggregation [13]. The benefit of this architecture, is that it gives the applications more information about network state from the controller and not only application-aware, compared to traditional networks where the network. The analysis of SND, architecture is based on [13] and the interested reader is referred to the original source.

More specifically, SDN architecture consists of three layers, namely the application layer, the control layer

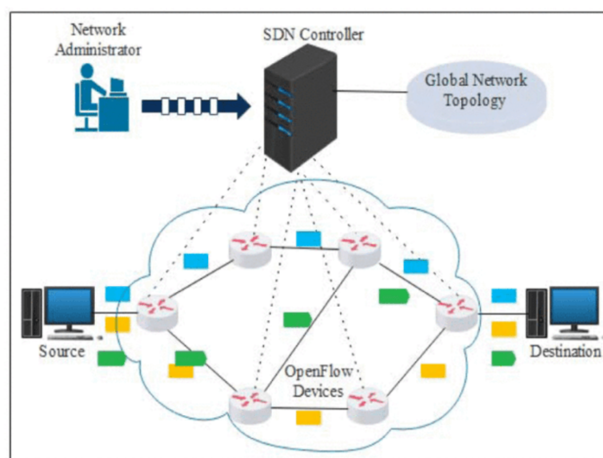


Figure 1: SDN Overview

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

and the infrastructure layer and are presented below [5] as shown in Figure 2.

Application Layer: SDN applications communicate behaviors and other needed resources with the SDN controller via application programming interfaces (APIs). On top of that, by collecting information from the controller, the applications can build an abstracted view of the network system. Such applications may be networking management, or enterprise applications used to run large data centers. For instance, an application for security purposes can be built in order to recognize suspicious network activity. Due to its nature, as it can dictates the behavior of the network, the application layer is also referred to as management plane and, as such, it also directs the controller to dynamically make decisions.

Control Layer: The control layer consists of a centralized SDN Controller, which acts as the logic of the network or its "brain" and lies in a server which contains the control logic. It is responsible for the management of network policies and the control of flow of traffic in the network. It also programs the infrastructure layer devices to transmit the traffic. Moreover, the SDN controller software extracts information about the network from the hardware devices and communicates back to the SDN applications with an abstract view of the network.

Infrastructure Layer: It consists of a set of networking equipment or hardware, such as switches, routers and middle-ware appliances in a network. The control plane dictates instructions and the infrastructure layer forwards packets. Due to its nature, it is also called data plane layer and is responsible for forwarding and processing of the data path.

As stated in [13], "the SDN architecture APIs are often referred to as northbound and southbound interfaces, defining the communication among the applications, controllers, and networking systems. A northbound interface is defined as the connection between the controller and applications, whereas the southbound interface is the connection between the controller and the physical networking hardware". It is because of the virtualized nature of architecture SDN, these elements do not have to be physically located in the same place.

2.3 SDN Applications

In this section several case examples, in which SDN is implemented are presented. The list is based on [14].

Hyper-Converged storage/Converged storage. Converged storage is a storage architecture consisting of computing resources and storage into an integrated framework and it can be used for development of platforms for storage or server centric or combined workloads. SDN begins to be used by a large number of well-known data services, for the creation of programmable technologies. The agility of keeping up the server is achieved by using virtual environments, which can as well virtualize the storage. For instance, Edgenet[1] (a globally distributed edge cloud) to create such a system have utilized a Programmable Flow SDN Ecosystem that makes use of the Openflow controller, along with strong and efficient switching to guarantee smooth and correct network effectiveness.

Video applications. A new application of SDN for video and collaboration applications is introduced by Sonus Networks (d/b/a Ribbon Communications[2]), which operates in real-time communications as a unified communications and control vendor. Their approach unifies the network virtualization platform of Juniper[3]

[1] https://edge-net.org/
[2] https://ribboncommunications.com/
[3] https://www.juniper.net/us/en/

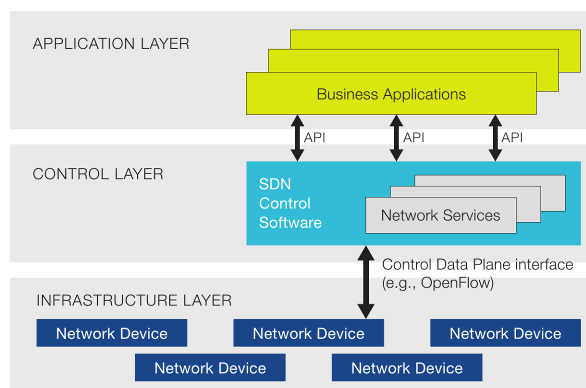


Figure 2: SDN Architecture

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

and their session border controllers, which results in great capacity creation for communication sessions. What is interested about their approach is that the use of SDN allows them to ensure the maintenance of the Quality of Service (QoS). On top of that, the usage of SDN gives the opportunity to assume control over the network and other pivotal agreements. Due to the fact that network space cab be allocated merely based on the need, overall effectiveness and capacity of the system can be further improved.

Orchestration of mobile network services SDN and as well as network functions virtualizations (NVFs) have both entered the mobile network operator (MNO) and many vendors have started to implement them in order to set up more robust networks in the bid, assuring both maximum resource utilization and dynamic provisioning. Using a combined framework of SDN and NVF, vendors can now provide the lead times. Apart from that, this unified framework provides companies with the unprecedented ability to separate their application logic. To conclude, catering the needs of vendors and an overall faster performance of platforms are possible through an efficient orchestration of services.

Data center networks' scalability. The wired infrastructure of data center networks can be replaced using SDN systems. Researches at the University of Illinois have investigated the usage of SDN switches to test a data center network. The scaling of bandwidth between servers can be guaranteed avoiding too much hardware cost. Many Pica8[4] switches have been installed consisting of hundered of ports. Access speeds are likely to be a lot higher as both load balancing and bandwidth scaling will be guaranteed.

Network of any enterprise. The extensive study of SDN advantages, such as better control and efficiency of operations, have made enterprise companies to switch to SDN as a network infrastructure and therefore many SDN vendors make use of a software operated networking system for business instead of the old physical networking infrastructure.

2.4 SDN Controllers

In a software-defined network, it is the SDN Controller that acts as a strategic control point and, hence, it is the "brain" of the network [15]. In order to deploy intelligent networks, "an SDN controller is used to manage flow control to the switches/routers 'below' (via southbound APIs) and the applications and business logic 'above' (via northbound APIs); they consolidate and mediate between different controller domains using common application interfaces" [15].

OpenFlow and Open Virtual Switch Database (OVSDB) are two of the most well-known protocols used by SDN controllers to communicate with the switches/routers. Many other controller protocols are being currently developed as for example the Internet Engineering Task Force (IETF) working group (the Interface to the Routing System (i2rs)) developed an SDN standard that enables a controller to leverage proven protocols, such as OSPF, and many others, across a variety of SDN platforms [15].

The overall architecture of the network can be influenced by the type of protocols [15]. For instance, OpenFlow operates in a different way as opposed to i2rs. The former attempts to totally, centralize packet-forwarding decisions, while the latter splits the decision making by leveraging traditional routing protocols to execute distributed routing and allowing applications to modify routing decision.

An SDN Controller platform consists of a collection of modules that can be plugged into the system and perform different network tasks [15]. Some of the basic tasks including inventorying what devices are within the network and the capabilities of each, gathering network statistics, and other monitoring functions, etc. Extensions can be inserted that enhance the functionality and support more advanced capabilities, such as running algorithms to perform analytics and orchestrating new rules throughout the network.

It is not mandatory that a controller platform and an application be, from the same company [15]. In this way, interoperability and flexibility are ensured. For instance, Cisco, offers an open-source controller built by OpenDaylight, being interoperable with many applications.

The first SDN controller was NOX, which was initially developed by Nicira Networks, alongside with OpenFlow and it has since become the basis for many SDN controllers. However, ONIX (an association of NTT, Google and Nicira Networks) the base for the Nicira/VMware controller is not open source [15].

On the other hand, there is a large number of open source controllers currently available. Next, a list of open and community driven initiatives of SDN controllers follows:

- Beacon
- Faucet
- ONOS
- Open vSwitch
- lighty-core
- Cherry
- NOX/POX
- OpenContrail
- RUNOS
- Ryu-Controller
- The Fast Data Project
- OpenDaylight
- OpenKilda
- Project Calico
- Project Floodlight
- vneio/sdnc

[4] https://www.pica8.com/

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

2.5 Challenges Associated with SDN

There are many challenges associated with SDN applications, some of which are discussed briefly below.

Reliability: Based on its definition, reliability of a software is stated as "...the probability of failure-free software operation for a specified period of time in a specified environment" [48]. Software reliability is of a great importance when it comes to software development. System developers must ensure the reliability and quality of their software, and should any failure comes up, following an automatic solution, the users should be informed. Hence, SDN, controller must be able should be prevent and accurately handle network errors. However, as SDN controller assumes total responsibility for the network, any failure may result in network collapse, and thus software reliability relies on the exploitation of the main controller functions[10].

Scalability: One definition[5] for Scalability states that "Scalability is an attribute that describes the ability of a process, network, software or organization to grow and manage increased demand." Other, definition for Scalability[6] states that "Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth". It is known that traditional LANs is, used in a multilayered architecture, in which multiple Layer 2 networks are connected using Layer 3 routing functionality. However, traditional LANs perform bad, in scaling when it comes to supporting east-west traffic, due to the fact that there are multiple Layer 3 devices lying in the end-to-end path [45]. It is stated that a minimum of 100 switches should be able to be supported by an SDN controller and also and impact of network broadcast overhead and the proliferation of flow table entries controller must addressed [45].

Low level interface: In case of SDN control applications for network management, SDN framework must translate the developed network policies into low-level configurations of the switch used [10]. Furthermore, in order to perform not so sophisticated, tasks, the multiple asynchronous events at the switches must be coordinated from the programming interface of SDN.

Performance and Security: Performance and security are two significant aspects when it comes to real world applications of SDN networks. It is said that new types of network attacks may emergence in open interfaces of such networks, resulting in performance reduction [10]. Malicious attempts, such as distributed denial-of-service (DDoS), may disrupt the normal traffic of networks. As such, several tools, including network threat detection and mitigation and authentication and authorization of the users among others, must be developed in order to avert any of the above disasters [10].

[5] https://www.techopedia.com/definition/9269/scalability

[6] https://www.networxsecurity.org/members-area/glossary/s/scalability.html

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

3 Network Functions Virtualization

3.1 Overview of NFV

Network Functions Virtualization (NFV) is referred to as the separating of network functions from hardware to create software in virtual machines (VMs) [16]. Virtual network functions (VNFs) may are different functions such as firewalls, traffic control, and virtual routing [16]. Infrastructure agnostic hardware is supported by NFV, which uses virtualized networking components, where virtualized resources of compute, storage, and as well as network functions can be placed on commercial off-the-shelf (COTS) hardware (as for example is x86, server) [16]. In that way, VMs can acquire available resources and multiple VMs can run on a single server and scale to consume the remaining free resources. What is interested, with this approach is that data centers with such virtualized infrastructure can be more effectively used, as the resources are not so much idle [16]. NFV can also virtualize data plane and control plane, within the data center and the outside networks. Figure 3 shows the high-level NFV framework and Figure 4 the original NFV architectural framework.

3.2 VNFs

The need to accelerate the deployment of new network services while reducing operating costs and capital expenses led to the development of Network virtualization and VNFs [50]. VNFs (virtual network functions), which handle specific network functions, such as firewalls, are at the core of NFV. Total, virtualized environment can be created using individual VNFs or a combination of them [50]. VNFs run on individual or multiple virtual machines (VMs) on top of the hardware networking infrastructure. In the latter case, where multiple VMs are on the hardware box, all of the box's resources is used [50].

To achieve the aforementioned goals of reducing cost, IT virtualization technologies are widely used. What is done by virtualizing a network is that both scale and diversity as well as the hardware cost are reduced. Only software is used, and should this software be updated, there is no need to change the whole hardware. In this way, many costly consequences have been addressed (for example, upgrading each piece of hardware, costs, required time and so on). Furthermore, virtualization helped the development of 5G technology, which will bring incalculable amounts of revenue to existing and new markets [50].

3.3 Network Services in NFV

A Network Function (NF) Forwarding Graph of interconnected NFs and end points can describe an end-to-end network service, such as mobile data, Internet access, among others [40].

From the architecture aspect, a network service can be viewed as a forwarding graph of NFs interconnected by supporting network infrastructure [40]. A single operator network can be used to implement NFs. On top of

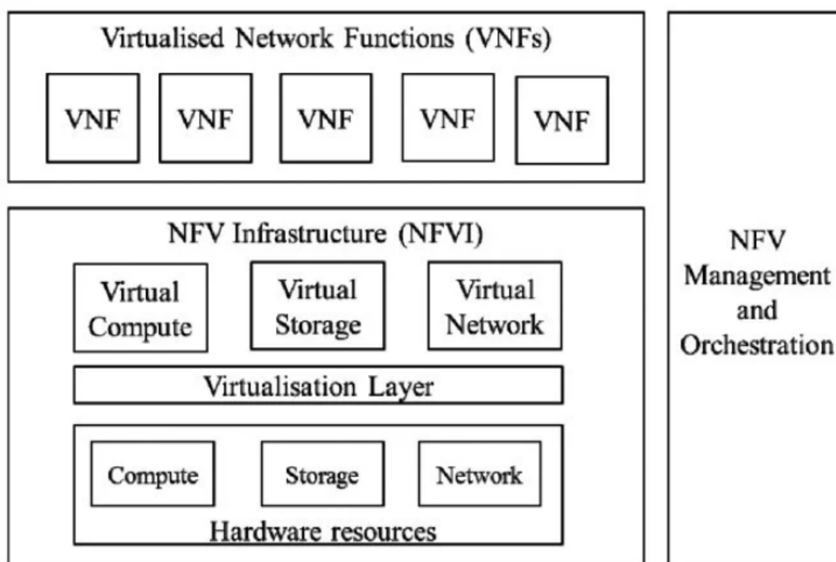


Figure 3: High-level NFV framework [16]

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

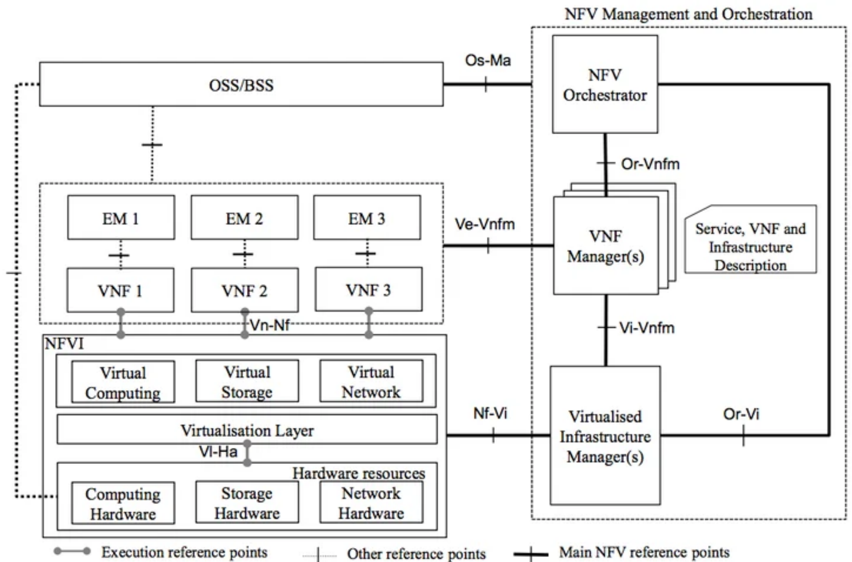


Figure 4: NFV original reference architectural framework [40]

that, an interwork between different operator networks is supported. These systems have particular, behavior, which is a combination of several functional blocks, such as NF forwarding graphs, sets to name but a few.

NFs and the end points are depicted as nodes and may refer to devices, applications and others. An NF Forwarding Graph can have network function nodes connected by logical links which can either be unidirectional or bidirectional or even multicast and/or broadcast [40]. A chain of network functions represent a plain example of forwarding graph and an end-to-end network service can include a smartphone, a wireless network. It is important to note that the NFV area of activity is within the operator-owned resources, and thus authority can be excersised in specific, domain (for example, a mobile phone out of the scope).

Figure 5 depicts an end-to-end network service. End Point A, the inner NF Forwarding Graph, and End Point B represent the outer end-to-end network service. Network functions NF1, NF2 and NF3 represent the inner NF Forwarding Graph. The latter are interconnected via logical links provided by the Infrastructure Network 2. As it can be seen, the end points (A and B) are connected to network functions via network infrastructure, which can either be wired or wireless. There is, logical interface (dot lines) between the end point and NFs.

Figure 6 depicts an example of an end-to-end network service. The different layers that are involved in its virtualisation process are presented as well. It is seen that this end-to-end network service consists of only VNFs and two end points. The virtualisation layer is responsible for the seperation of hardware and software in NFV. NFVI-PoPs include several resources dedicated for computation, storage and networking.

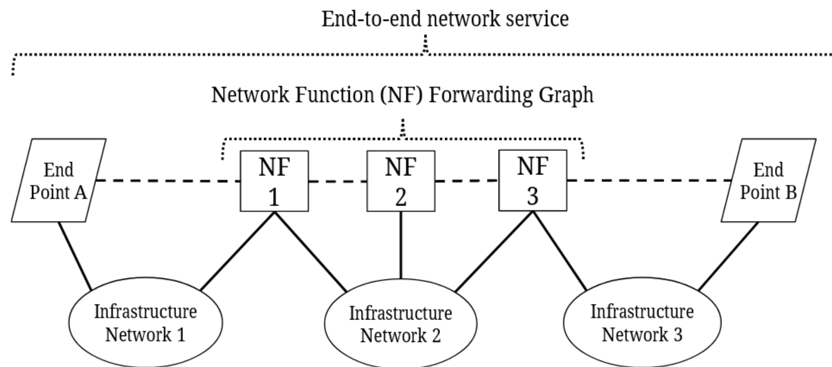


Figure 5: Graph representation of an end-to-end network service [40]

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

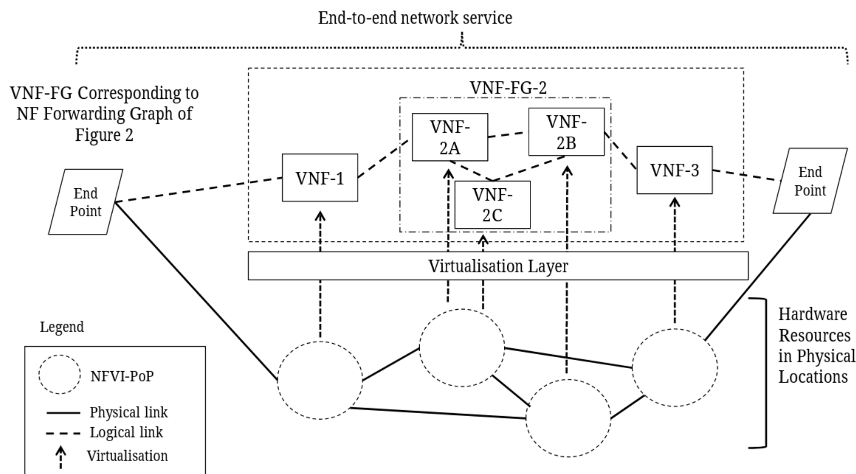


Figure 6: End-to-end network service [40]

3.4 The Benefits of NFV

There is a plethora of benefits when it comes to NFV. Some of the benefits are listed and briefly described below [40]:

- **Improved capital efficiencies**: achieved by using commercial-off-the-shelf (COTS) hardware and results in improved capital efficiencies when compared to dedicated hardware implementations.
- **Improved flexibility**: which fosters assigning VNFs to hardware, which in turn aids both scalability and largely decouples functionality from location.
- **Rapid service innovation**: achieved via software-based service deployment.
- **Improved operational efficiencies**: this stems from automating and operating practices.
- **Reduced power usage**: this is achieved by transferring workloads and switching off not used hardware.
- **Standardized and open interfaces**: this results in providing decoupled elements by different vendors.

3.5 Challenges Associated with NFV

NFV confronts various challenges center around the three components of NFV, namely the NFV manager (NFVM), the NFV infrastructure (NFVI) and VNFs [16]. It is because of how these pieces are built and how they are interwoven that adds complexity to the system when deploying NFV at scale. Due to the fact that there are many approaches in the whole industry and no standars, have been set, is complexity in the components of NFV technology.

3.6 NFV vs SDN

When it comes to comparing NFV and SDN technologies, one should notice that NFV refers to the virtualization of network components, whereas SDN refers to a network architecture that provides the network with programmability. The former, virtualizes network infrastructure, while the latter centralizes network control. SDN and NFV can be combined into an integrated framework in order to create a fully operated and manageable network by software [16].

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

4 Programming Protocol-Independent Packet Processors

4.1 Overview

Configuring and managing large scale networks are fostered using SDN; on the other hand, it is common that SDN control protocols limit the possibilities for the aforementioned targets, they provide only a limited set of protocols, which are not understood by a fraction of the available hardware [8]. Thus, a new technology comes to address this issue. This technology, known as Domain-Specific Language (DSL) P4, is designed in such a way that a brief description of a switch's behavior is allowed. The aim behind this approach is to give owners the opportunity to develop their own software-oriented application for a programmable switch. P4 is translated into an executable program, using either a compiler or an interpreter [8]. Figure 7 depicts, an P4 architecture.

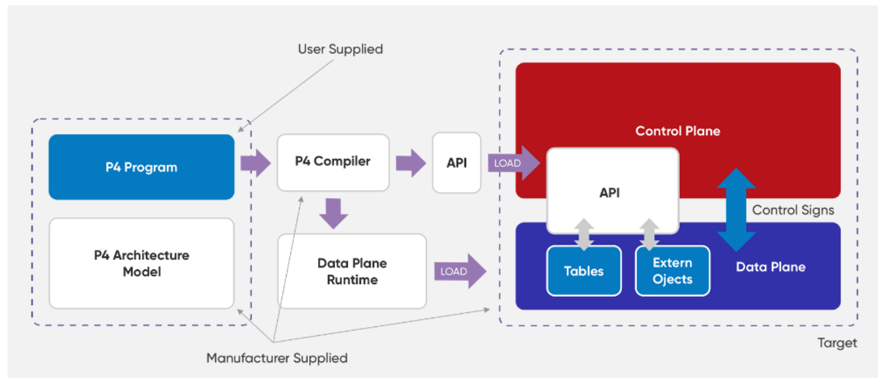


Figure 7: P4 Architecture

It is interesting to mention the design goals of P4, which are given briefly below [8]:

- *Reconfigurability in the field.* Since switch behavior changes, modification of its behavior must remain possible.
- *Protocol independence.* P4 is not restricted in the assumption regarding the used protocols, rather it can define and integrate new protocols formats whenever necessary.
- *Target independence.* P4 programs can be used on any hardware, given a runnable translation of P4.

The motivation behind the development of P4 lies in the fact that OpenFlow is unable to use custom protocols. However, P4 borrows from OpenFlow some advantages regarding packet processing [8]. Packets are processed by performing actions based on values of the header fields and as in OpenFlow the mapping is done during runtime by a control plane instead of at compile time.

Processing packets in P4 consists of four major phases [8]:

1. *Parsing of the packet.* Once a packet is received, the first thing to do is to translate it into a representation to be processed in the subsequent phases. The underlying P4 program generates a finite state machine to base the parse on.
2. *Apply match-action table to ingress.* In this phase, the received object now enters the ingress pipeline. There is no restriction concerning the execution of rules on the received packet, as it is possible to match on the different header fields. Furthermore, the switch can decide from which egress pipeline the packet will be processed later. P4 program can access additional information, such as on which hardware port the packet arrived (packets can be resubmitted to enter the ingress pipeline, if need be).
3. *Apply match-action table to egress.* Egress pipeline allows execution of rules based on the parsed header fields (submission to another egress pipeline or resubmits cannot be used).
4. *Deparsing of the packet.* The last phase is about packet deparsing based on its current state, so it can be sent to the wire. The parsed object automatically generates the deparser.

Figure 8 depicts which phases are affected by which program parts. As mentioned, P4 program statements affects different parts of the processing pipeline.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

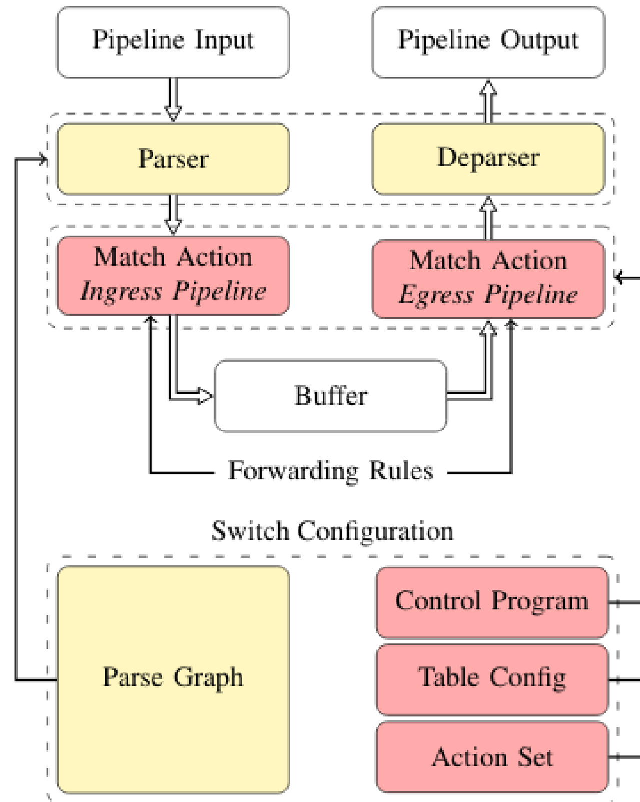


Figure 8: Packet processing as done in P4 [8]

4.2 Behavioral Model

As mentioned in the previous subsection, in order to receive an executable file from a P4 program either a compiler or an interpreter is required. To that effect, the P4 Language Consortium published the P4 compiler p4c-behavioral [34]. There are many reasons [8] that p4c-behavioral is replaced by the "behavioral model" bmv2 [32]. Written in C implements all features of the P4 specification. P4, program is the input and the output is a valid C program. In order to achieve the previous, p4-hlir, a program to create a High-Level Intermediate Representation of P4 is used. p4c-hlir [35], written in Python, provides a target independent P4 parser. Python object hierarchy is available after a successful parse. p4c-behavioral, using p4-hlir, generates correct C code, which in turn can be complied for the intended target, whereas C compiler can further optimize the process. The Apache 2 licensed source code can be found at [34].

The fact that double compilation is needed (from P4 to C and from C to binary) in order to produce an executable is rather a difficult task. On top of that, the generated C code from p4c-behavioral is extremely difficult to comprehend. Furthermore, the presence of two pipelines (an ingress and an egress) is assumed, which contradicts the paradigm that P4 does not require any hardware properties; changes are needed in order to fully support P4.

A new behavioral model [32], also called bmv2, was developed. The behavioral model is now an interpreter. In order to run a P4 program, first a compilation of the P4 source code to a JSON file is needed and combined with the P4 files, it represent the input for for the interpreter. It is noted that the JSON output is generated by p4c-bm [37], which also can generate program dependent C++ code [8]. On the switch, to enable communication between the control plane and the forwarding plane, the program dependent code is either an Apache Thrift [42] or an nanomsg [31] based mechanism. The new interpreter is released under the Apache 2 license. It fully supports the P4 specification and can be integrated into mininet [44].

In order for one to to run their own P4 programs in bmv2, first, they need to compile the P4 code into a JSON representation. This representation will inform bmv2 to initialize specific tables, to configure the parser, the checksums, the ingress and the egress and as well as the deparser.

There are currently two P4 compilers available for bmv2 on p4lang:

- p4c includes a bmv2 backend and is the recommended compiler to use. At the moment, the bmv2 p4c

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

backend supports the v1model architecture, with some tentative support for the PSA architecture. P4 programs written for v1model can be executed with the simple_switch binary, while programs written for PSA can be executed with the psa_switch binary.

- p4c-bm is the legacy compiler for bmv2, which is no longer actively maintained.

4.3 V1model

4.3.1 Architecture

The V1Model architecture is implemented on top of the bmv2's simple_switch target. P4.org developed a software switch named bmv2 simple_switch for testing the functionality of P4 programs. There exist other architectures that can be supported by bmv2, however V1Model is the one that is used in this thesis. As shown in Figure 9, V1Model consists of a P4 programmable parser and deparser, blocks to verify and update checksums on incoming and outgoing packets, respectively, separate ingress and egress pipelines of match-action processing, as well as a traffic manager. The traffic manager is not currently programmable in P4 but performs the task of scheduling and sometimes replicating packets between the input and output ports. The architecture description also includes a set of standard metadata fields that are used by the P4 program to direct the packet through the bmv2 simple_switch.

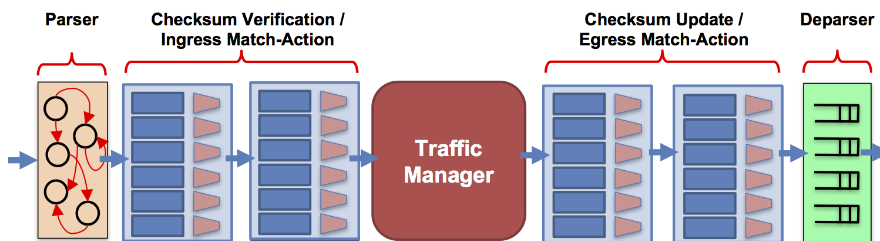


Figure 9: V1model Architecture

4.3.2 Standard Metadata

On the code snippet below is a list of the various standard metadata fields defined for the V1Model. The most commonly used fields are:

- The `ingress_port` is set by the simple_switch target and indicates the port on which the packet arrived
- The `egress_spec` field should be set in the ingress pipeline to tell the traffic manager which port to send the packet to
- The `egress_port` can be read within the egress pipeline, and indicates which port the packet is departing from

There are also many other fields here that can be used by P4 programs that target the V1Model.

```
struct standard_metadata_t {
   bit<9>  ingress_port;
   bit<9>  egress_spec;
   bit<9>  egress_port;
   bit<32> clone_spec;
   bit<32> instance_type;
   bit<1>  drop;
   bit<16> recirculate_port;
   bit<32> packet_length;
   bit<32> enq_timestamp;
   bit<19> enq_qdepth;
   bit<32> deq_timedelta;
   bit<19> deq_qdepth;
   bit<48> ingress_global_timestamp;
   bit<32> lf_field_list;
   bit<16> mcast_grp;
   bit<1>  resubmit_flag;
   bit<16> egress_rid;
   bit<1>  checksum_error;
}
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

4.3.3 Program Template

A template for a typical P4 program written for the V1Model is given below. As shown, core.p4 must be included at the top of the file, because it defines some of the common types that are used in all P4 programs. v1model.p4, should also be included, which provides the definition of the V1Model architecture and also declares the various externs, such as counters, meters, and registers, that can be used for this target. A P4 program typically begins by declaring the format of various packet headers and building the headers and metadata structs. These structs are passed from block to block, along with the standard_metadata, as the packet traverses the architecture. Subsequently, the P4 source file (or files) must define the functionality for the following:

- parser
- checksum verification block
- ingress match-action processing
- egress match-action processing
- checksum update block
- deparser

And finally, the P4 program must instantiate an instance of the architecture called main.

```
1  #include <core.p4>
2  #include <v1model.p4>
3  /* HEADERS */
4  struct metadata { ... }
5  struct headers {
6      ethernet_t  ethernet;
7      ipv4_t      ipv4;
8  }
9  /* PARSER */
10 parser MyParser(packet_in packet,
11         out headers hdr,
12         inout metadata meta,
13         inout standard_metadata_t smeta) {
14   ...
15 }
16 /* CHECKSUM VERIFICATION */
17 control MyVerifyChecksum(in headers hdr,
18                          inout metadata meta) {
19   ...
20 }
21 /* INGRESS PROCESSING */
22 control MyIngress(inout headers hdr,
23                   inout metadata meta,
24                   inout standard_metadata_t std_meta) {
25   ...
26 }
27 /* EGRESS PROCESSING */
28 control MyEgress(inout headers hdr,
29                  inout metadata meta,
30                  inout standard_metadata_t std_meta) {
31   ...
32 }
33 /* CHECKSUM UPDATE */
34 control MyComputeChecksum(inout headers hdr,
35                           inout metadata meta) {
36   ...
37 }
38 /* DEPARSER */
39 control MyDeparser(inout headers hdr,
40                    inout metadata meta) {
41   ...
42 }
43 /* SWITCH */
44 V1Switch(
45     MyParser(),
46     MyVerifyChecksum()
47     MyIngress(),
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
48     MyEgress(),
49     MyComputeChecksum(),
50     MyDeparser()
51 ) main;
```

4.3.4 Basic Example

The code snippet below illustrates a more basic example of a P4 program. It is essentially just a wire that connects port 1 to port 2. For this example, the parser does not need to extract any headers from the packet; it simple accepts and the packet is passed along to `MyVerifyChecksum()` block, which also does not need to do anything. When the packet reaches the `MyIngress()` block, the P4 program checks the port on which the packet arrived by reading the ingress_port field of the standard_metadata bus and decides which port to send the packet to using a simple `if`, `else-if` statement. The rest of the blocks do not need to do anything and at the end of the program the switch is instantiated. As it can be seen, all the packets that arrive on port 1 will be sent to port 2 and all the packets that arrive on port 2 will be sent to port 1. In this example, the connection between port 1 and port 2 is hardcoded and cannot be changed after the P4 program is compiled.

```
1  #include <core.p4>
2  #include <v1model.p4>
3  struct metadata {}
4  struct headers {}
5
6  parser MyParser(packet_in packet,
7                  out headers hdr,
8                  inout metadata meta,
9                  inout standard_metadata_t standard_metadata) {
10
11     state start { transition accept; }
12 }
13
14 control MyVerifyChecksum(inout headers hdr, inout metadata meta) { apply { } }
15
16 control MyIngress(inout headers hdr,
17                   inout metadata meta,
18                   inout standard_metadata_t standard_metadata) {
19
20     apply {
21         if (standard_metadata.ingress_port == 1) {
22             standard_metadata.egress_spec = 2;
23         } else if (standard_metadata.ingress_port == 2) {
24             standard_metadata.egress_spec = 1;
25         }
26     }
27 }
28
29 control MyEgress(inout headers hdr,
30                  inout metadata meta,
31                  inout standard_metadata_t standard_metadata) {
32     apply { }
33 }
34
35 control MyComputeChecksum(inout headers hdr, inout metadata meta) {
36      apply { }
37 }
38
39 control MyDeparser(packet_out packet, in headers hdr) {
40     apply { }
41 }
42
43 V1Switch(
44    MyParser(),
45    MyVerifyChecksum(),
46    MyIngress(),
47    MyEgress(),
48    MyComputeChecksum(),
49    MyDeparser()
50 ) main;
```

The switch can be made a bit more flexible. This design achieves the exact same functionality as the above, but rather than using the `if`, `else-if` statement in the MyIngress control block, it uses a single table, called forward. The forward table matches on the `ingress_port` field, and can invoke either the

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

set_egress_spec action, or NoAction. The table entries specify which action to take, along with any input parameters, for different ingress_port values. Writing the P4 program in this way, a much more flexible switch can be created, because now table entries can be added and removed from the control plane to determine how to route packets between the various ports. And this can be done without changing the P4 program at all.

```
#include <core.p4>
#include <v1model.p4>
struct metadata {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {
    state start { transition accept; }
}

control MyIngress(inout headers hdr, inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action set_egress_spec(bit<9> port) {
        standard_metadata.egress_spec = port;
    }
    table forward {
        key = { standard_metadata.ingress_port: exact; }
        actions = {
            set_egress_spec;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }
    apply { forward.apply(); }
}

control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply { }
}

control MyVerifyChecksum(inout headers hdr, inout metadata meta) { apply { } }

control MyComputeChecksum(inout headers hdr, inout metadata meta) { apply { } }

control MyDeparser(packet_out packet, in headers hdr) { apply { } }

V1Switch(
   MyParser(),
   MyVerifyChecksum(),
   MyIngress(),
   MyEgress(),
   MyComputeChecksum(),
   MyDeparser()
) main;
```

4.3.5 Basic and Header Types

The basic types that can be used in P4 programs are unsigned and signed integers. An unsigned integer is called a "bitstring" in P4 and can have an arbitrary width; many different operations on bitstrings, such as addition, subtraction, and concatenation can be performed. Also, one can slice bitstrings similar to how they might slice a python string or a verilog bus. Signed integers are simply called ints in P4 and they support some, but not all, of the same basic operations as bitstrings. P4 also supports the notion of a variable sized bitstring for protocols that contain fields whose width is only known at runtime. For example, IPv4 options. In this case, n is the maximum possible width of the bitstring. The P4 specification [33] have a list of with all the possible operations that are supported for these types.

A header is an ordered collection of members each of which may be one of the types just presented. Headers are byte-aligned and they can be either valid or invalid. P4 programs can read and write a header's validity bit using the isValid(), setValid(), and setInvalid() methods. The following snippet showing how you might declare an Ethernet and IPv4 header in P4. The usage of the typedef statement can make the code

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

more readable.

```
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;
header ethernet_t {
  macAddr_t r;    dstAdd
  macAddr_t srcAddr;
  bit<16>   etherType;
}
header ipv4_t {
  bit<4>    version;
  bit<4>    ihl;
  bit<8>    diffserv;
  bit<16>   totalLen;
  bit<16>   identification;
  bit<3>    flags;
  bit<13>   fragOffset;
  bit<8>    ttl;
  bit<8>    protocol;
  bit<16>   hdrChecksum;
  ip4Addr_t srcAddr;
  ip4Addr_t dstAddr;
}
```

4.3.6 Other Types

The `struct` is another useful type. It consists of an unordered collection of members and has no alignment restrictions. An example use case of a `struct` has already been presented, where the standard_metadata for the V1Model was presented. P4 programs often combine a collection of headers into a `struct` and then use that `struct` within each programmable block. In addition, multiple headers of the same type can also be combined into a header stack, which is essentially treated as an array of headers. A header union represents an alternative containing at most one of several different headers. For example, one might want to declare a header union consisting of an IPv4 header and a IPv6 header if they expect all the packets to contain only one of these and never both.

```
/* Architecture */
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    ...
}

/* User program */
struct metadata {
    ...
}
struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
}
```

4.3.7 Parser

Next, the steps needed to write parsers in P4 are presented. Parsers are functions that map packets into headers and metadata. They are written as a state machine that contain 3 predefined states: start, accept, and reject. It may also contain other states defined by the P4 programmer and it is common to define one state for each header type that the parser will extract, but this is not required. Parsing always starts in the start state, executes zero or more statements, and then transitions to another state until it encounters either the accept or reject state. It is up to the architecture to decide what should happen upon reaching the reject state. For the V1Model, the packet is not dropped. Furthermore, loops are acceptable within the parsing state machine.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
/* From core.p4 */
extern packet_in {
    void extract<T>(out T hdr);
    void extract<T>(out T variableSizeHeader,
                    in bit<32> variableFieldSizeInBits);
    T lookahead<T>();
    void advance(in bit<32> sizeInBits);
    bit<32> length();
}
/* User Program */
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t std_meta) {

    state start {
        packet.extract(hdr.ethernet);
        transition accept;
    }
}
```

The parser in the V1Model accepts the input packet and must produce the parsed header representation. The user metadata and standard metadata buses also pass through the parser and can be used if desired. In lines 2-9 of the snippet depicted above, the definition of the packet_in type from core.p4 is shown. This type has various methods that are very useful when writing the parser. In this example, only the most basic extract method is used, which allows extraction of a fixed size header from the packet. The language specification [33] contains more details on the other more advanced parsing methods. Additionally, in lines 11-20 an example of a very simple parser is given that only defines the functionality of the start state; it simply extracts the ethernet header from the packet and then transitions to the accept state.

To design parsers with more states, the `select` statement is often used. The structure is very similar to a case statement in C or Java, but without the "fall-through behavior", meaning that the use of break statements is not needed. Parsers commonly want to branch based on some of the bits that were just parsed. For example, on the snippet bellow a parser that extracts the ethernet header and then branches to either the parse_ipv4 state or the accept state based on the ethernet's etherType field is shown. The match patterns can either be literals or simple computations, such as masks.

```
state start {
    transition parse_ethernet;
}

state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {
        0x800: parse_ipv4;
        default: accept;
    }
}
```

4.3.8 Simple Actions

Aside from parsers, control blocks are the only P4 programmable elements in an architecture. These blocks are similar to C functions, but do not contain any loops. Variables, tables, instantiate externs, and more can be declared. The actual functionality of a control block is specified within the apply block. All kinds of processing that are expressible can be represented as a directed acyclic graph, including: match-action pipelines, deparsers, and additional forms of packet processing. The interfaces between a control block and the other blocks in an architecture are defined by user and architecture defined types, typically headers and metadata.

In the following snippet a very simple example of a control block is shown that swaps the source and destination MAC addresses and then sends the packet back out the same port on which it arrived. A 48-bit variable called tmp is defined to hold the current destination MAC address while the swapping is taking place. This temporary variable is local to this particular control and cannot be accessed by other blocks. The temporary variable could also have been declared inside the `apply` statement to achieve the same functionality.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
    /* Declarations region */
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
    bit<48> tmp;

    apply {
        /* Control Flow */
        tmp = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp;
        std_meta.egress_spec = std_meta.ingress_port;
    }
}
```

The previous example can also be implemented by defining a simple action called swap_mac and then invoking the action directly within the `apply` statement, similar to how one might invoke a C function. These actions can be declared either inside a control block or globally. Since this action is invoked directly within the `apply` statement, each action parameter must specify both type and direction. Actions may contain local variables and may make use of many different standard arithmetic and logical operations, as shown here.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {

    action swap_mac(inout bit<48> src,
                    inout bit<48> dst) {
        bit<48> tmp = src;
        src = dst;
        dst = tmp;
    }

    apply {
        swap_mac(hdr.ethernet.srcAddr,
                 hdr.ethernet.dstAddr);
        std_meta.egress_spec = std_meta.ingress_port;
    }
}
```

4.3.9 Tables

Tables are the fundamental unit of a match-action pipeline. A table declaration will specify a number of different things including what data to match on as well as the type of match to be performed, a list of all possible actions and it may optionally specify a number of different table properties such as the number of entries, a default action to invoke if no match is found, or static table entries. Each table contains one or more entries, commonly referred to as rules. A single entry contains a specific key to match on, a single action that is executed when a packet matches the entry and any data to provide to the action when it is invoked, which may be nothing.

Steps involve performing match-action processing. As just mentioned, our table consists of a number of entries that specify a key, an action ID, and action data. These entries can be added and removed by the control plane. A table will also specify a default action to perform if no match is found amongst the table entries. The table will extract the specified bits from the input headers and metadata to form the key and attempt to find a match within its entries.

- if there is a hit then the action ID and data are read out of the corresponding entry
- if there is not a hit then the default action ID and data are used

These two outputs are passed into a multiplexer which selects one of the two based on whether or not a hit was found in the table. The chosen action ID and data are passed to the action execution unit along with the input headers and metadata which then updates the headers and metadata as necessary. Figure 10 shows the tables match-action processing.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

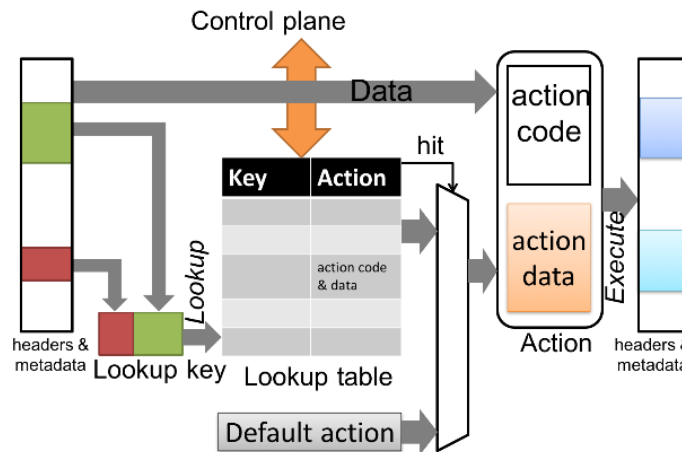


Figure 10: Match-Action Unit Dataflow

For instance, if we want to build a simple IP router, a definition of a routing table is required. The routing table must perform a longest prefix match on the destination IP address. Each table entry may invoke either the ipv4_forward action which updates the packet's destination MAC address, configures the packet's egress port, and decrements the TTL, or it may invoke the drop action. The default will be to perform no action at all.

In order to make IP routing working properly, the data plane and control plane must work together. The P4 **data plane** defines the format of the routing table, it specifies fields to match on and possible actions to invoke. It performs the table lookup and executes the chosen action. The **control plane** is responsible for populating the table with entries. There may potentially be many different control mechanisms for manipulating table entries. For example, network operators' manual configuration, automatic discovery, or routing protocol calculations.

This is what the routing table -that just described- would look like in P4. Here, the table `ipv4_lpm` is called. The key is the IPv4 destination address and a match type of lpm is specified, which stands for longest prefix match. The second table property is the list of all possible actions that can be invoked by this table. In this case, the possible actions are called `ipv4_forward`, `drop`, and `NoAction`. The table is also specified and should be allocated with enough room for 1024 entries and the default_action should be `NoAction`.

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}
```

Tables can specify various different match types, core.p4 defines 3 types that all architectures should support: exact match, ternary match, and longest prefix match. Other architectures may define additional match types that can be used in P4 programs. For example, the V1Model defines a range type and a selector type match in addition to the three standard types.

```
/* core.p4 */
match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */
match_kind {
    range,
    selector
}

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
/* Some other architecture */
match_kind {
    regexp,
    fuzzy
}
```

It has already been described how one can define actions and invoke them directly from a control block's `apply` statement. However, actions can be invoked from tables as well. Actions can have two different types of parameters, corresponding to these two ways of invoking actions **directional** and **directionless** parameters. Directional parameters are those that are passed from/to the data plane and directionless parameters are those are provided by table entries, in other words, from the control plane. Moreover, actions that are called directly only use directional parameters and actions that are used in tables typically use directionless parameters, but may also use directional parameters. In the snippet below, the declarations for the actions of a simple IP router is shown. Note that the parameters for the `ipv4_forward` action are directionless because it will be invoked from the routing table.

```
/* core.p4 */
action NoAction() {
}

/* basic.p4 */
action drop() {
    mark_to_drop();
}

/* basic.p4 */
action ipv4_forward(macAddr_t dstAddr,
                    bit<9> port) {
    ...
}
```

The table, after its definition, must be applied within the control flow. This is done by invoking the table's apply() method as shown here.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    table ipv4_lpm {
        ...
    }
    apply {
        ...
        ipv4_lpm.apply();
        ...
    }
}
```

4.3.10 Deparser

A control block can also be used to implement the deparser. Deparsing involves assembling the headers back into a well-formed packet. Use the emit method for the `packet_out` extern to insert a specific header into the final packet if it is valid.

```
/* From core.p4 */
extern packet_out {
    void emit<T>(in T hdr);
}

/* User Program */
control DeparserImpl(packet_out packet,
                     in headers hdr) {
    apply {
        ...
        packet.emit(hdr.ethernet);
        ...
    }
}
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

4.4 P4 Runtime

A novel way to control the forwarding plane of a switch, router, firewall and others is the P4 Runtime [17]. Among other advantages, P4 Runtime permits the control of any forwarding plane, regardless of whether or not it is built from a fixed-function or programmable switch (for example, ASIC) or a software switch running on an x86 server. Protocols and features the forwarding plane supports do not effect the framework of P4 Runtime, and therefore a huge variety of different switches can be controlled using the same API. If it is the case where new protocols and features are added to the forwarding plane, the P4 Runtime API is automatically updated. This is done by describing how a new feature must be controlled (note that this is achieved without restarting or rebooting the control plane). P4 Runtime is unaware where the control plane lies. The latter could either be a protocol stack running on a local switch operating system or a remote control plane running on x86 servers.

There are a lot of issues concerning closed and fixed APIs [17]. Such APIs are written to cover the needs of specific switch chip and seldom extensions are needed over time. Apart from that, due to the fact that these APIs are proprietary, further problems concern non-disclosure and license agreements, which prohibit sharing the API.

However, significant attempts have been made towards replacing closed APIs with open interfaces [17]. The first try concerns the OpenFlow, released a decade before, allowing a remote control plane to control switches from different vendors using the same API and other advantages. On the other hand, there are many issues concerning OpenFlow. First things first, it was design to fit on a certain use cases and fixed function switches. On top of that, it was not designed to be extended and its overall behavior was unclear. These peculiar characteristics of OpenFlow lead the way to other solutions.

SAI [49] represents such a solution, but it is restricted to be used for networks in which the control plane is inside the switch. SAI can be used to control switches based on a variety of different switch ASICs, just as OpenFlow could do. SAI becomes more complex over time, its extension is rather a cumbersome task, while the remote control of switches is ambiguous [17]

P4 Runtime comes to address the aforementioned issues [17]. P4 Runtime is (i) open and can be used to control any switch ASIC, (ii) extensible as new features can be easily added over time, (iii) customizable as different protocols and features can be used by different networks using the same API. Like SDN, P4 Runtime can either control switches from a remote control plane or from a local control plane running on the same switch. It is important to note that defining the forwarding behavior of both protocols in P4 [38] P4 Runtime can emulate both the behavior of OpenFlow and SAI; in that way, can be easily extensible over time [17].

Figure 11 depicts P4 Runtime API can be used to control a switch with remote control plane. The P4 program "tor.p4" specifies the switch pipeline and a P4 compiler generates the schema needed by the P4 Runtime API to add and delete entries into the forwarding table at runtime.

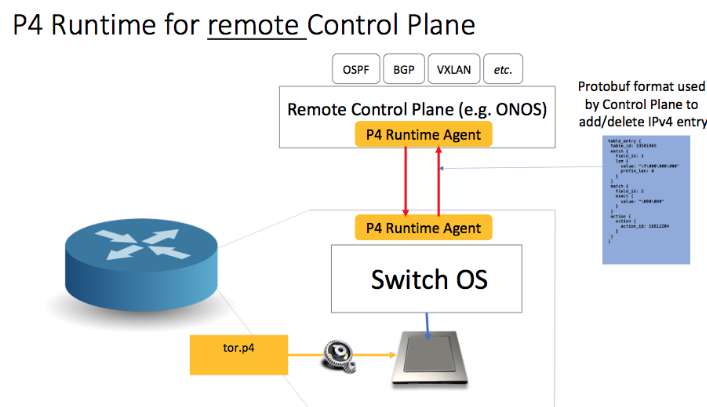


Figure 11: Using P4 Runtime API with a remote control plane [17]

Figure 12 illustrates the case where a local control plane can use P4 Runtime as an API.

Any switch, whose behavior is specified in the P4 language, can be controlled by the P4 Runtime API [17]. A developer can use P4 Runtime to control existing fixed-function switches by first writing a P4 program to document the switch behavior using the P4 language. A P4 compiler, p4c, automatically identifies elements that need to be controlled, such as lookup tables specified in the P4 program for which entries need to be added and deleted.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

P4 Runtime for local Control Plane

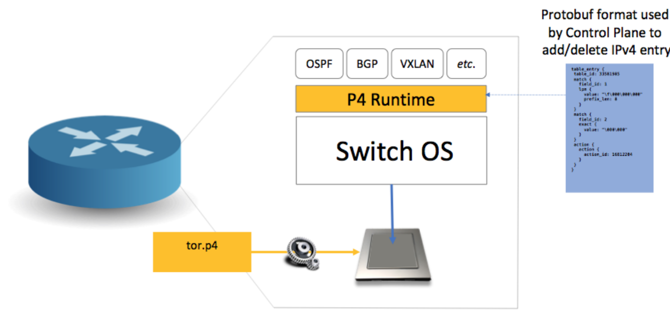


Figure 12: Using P4 Runtime API with a local control plane [17]

An example from [17] is adopted to illustrate the previous point. Given Figure 13, the P4 program includes an IPv4 longest-prefix-match (LPM) table. From this table, entries will need to be added and deleted, such as the 8-bit prefix (middle) when the switch is running. Then, the compiler generates the general schema (right). The protobuf [43] schema is used by the control plane to encode the specific 8-bit prefix that it wants to add to the table.

Note that new tables may be added that can be controlled through the P4 Runtime API (such as an IPv6 prefix table, or even a new table private to the network) given a programmable switch; the compiler will extend the schema (protobuf message), and then new tables can be controlled [17].

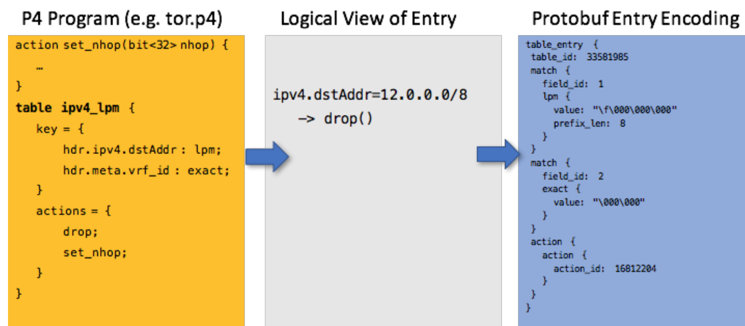


Figure 13: P4 runtime dataflow [17]

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

5 Open Network Operating System

5.1 Overview of ONOS

In case where next-generation SDN/NFV solutions are needed, Open Network Operating System (ONOS) is deemed as the leading open source SDN controller [18]. ONOS is an SDN controller open source project that uses OSGI technology, to manage sub-projects [19]. Using ONOS, both configuration and real-time control of the network are supported. As a result, the need to run routing and switching control protocols inside the network fabric is eliminated. ONOS, cloud controller, fed with intelligence, new network applications can be easily created avoiding, any need to alter the data plane systems. Due to its flexibility, with ONOS new dynamic network services with simplified programmatic interfaces are possibly.

The ONOS platform includes [18]: a platform and a set of applications acting as an extensible, modular and distributed SDN controller, simplified management, configuration and deployment of new software, hardware and services and a scale-out architecture, which provides both the robustness and scalability, which are required to when it comes to strict production carrier environments

5.2 ONOS Design

Design goals regarding the initial design of ONOS include [19]:

- **Code modularity**: where new functions as a new independent unit are supported
- **Features are configurable**: where dynamic loading and unloading features are supported, regardless startup or runtime phase
- **Protocol-independent**: where specific protocol libraries and implementations do not restrict the applications

As stated above, ONOS can manage sub-projects, each of which has its own source code tree, built independently. For that purpose, ONOS's source code is organized in a hierarchical way to make easier the use of Maven's cascaded POM file organization [19]. On top of that, each sub-project has its own pom.xml file and directory. The shared dependencies and configuration of the parent Pom file are inherited from the sub-pom.xml file and they can be built independently of other independent sub-projects. The top-level POM files used to build a complete project as well as its modules are contained in the root directory.

ONOS uses Karaf as its OSGI technology and some of the features that Karaf support are listed below [19]:

- **use of standard JAX-RS API** to develop a secure API interface
- **features for centralized custom settings** (as a set of Bundles)
- **local and remote SSH console login** through an easy-to-expandable command-line framework
- **records of different log levels**

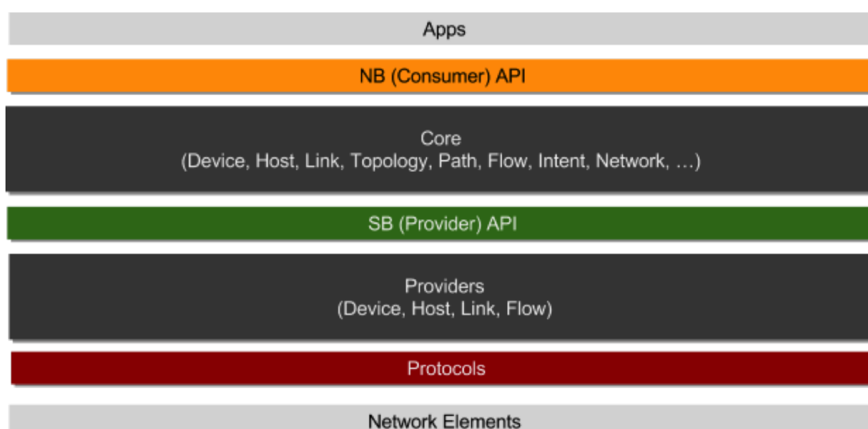


Figure 14: ONOS stack [19]

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

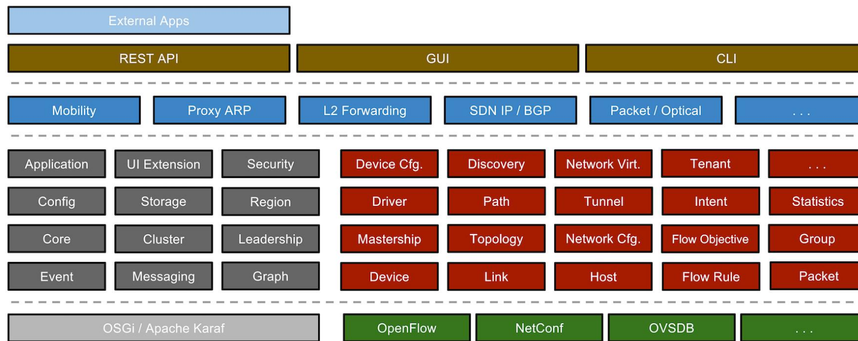


Figure 15: ONOS components [19]

ONOS, independently from the protocol, consists of the below parts, coming in layered architecture [19]: protocol awareness module that interacts with the network, Protocol-independent system Core, tracking and serving network status information, applications that consume and operate based on system information provided by Core.

As Figure 14 depicts, the Core interacts with the network-oriented modules through a southbound (provider) API and with the application through the northbound (consumer) API. Note that southbound API defines a protocol-neutral means to pass network status information to the Core, where the latter interacts with network devices through network-oriented modules. Note again that the Northbound API provides applications with abstractions describing network components and attributes; this is done so they can define their required actions based on policies [19].

5.3 ONOS System Components

The collection of components that make up a service is called a **subsystem**, while a **service** is referred to as a functional unit consisting of multiple components at different layers as a software stack to create vertical slices [19]. Figure 15 depicts the subsystems included in ONOS. Below ONOS's primary services are given [19]:

- ***Device Subsystem***, which manages the inventory of infrastructure devices
- ***Link Subsystem***, which manages the inventory of infrastructure links
- ***Host Subsystem***, which manages the inventory of end-station hosts and their locations on the network
- ***Topology Subsystem***, which manages time-ordered snapshots of network graph views.
- ***PathService***, using the most recent topology graph snapshot, computes the paths between infrastructure devices or between end-station hosts
- ***FlowRule Subsystem***, which manages the inventory of the match/action flow rules installed on infrastructure devices and provides flow metrics
- ***Packet Subsystem***, which allows applications to consider data packets received from network devices and to emit data packets into the network through one or more network devices

5.4 Subsystem structure

Figure 16 depicts the relationship between the subsystem components (note that the dotted lines indicate the boundaries between layers created by the north and south APIs, respectively). A detailed description about the architecture illustrated in Figure 16 is given in [19]. The main levels can be identified by one or more interfaces implemented by Java [19].

The Provider component lies in the bottom layer of the stack. The communication of `Provider` interface with the underlying device is achieved through a protocol-specific library. The interaction with core is done via the `ProviderService` interface. Note that a specific `ProviderId` is related to a Provider and its purpose is to provide an external identity of the `Provider` family, needed so that the Provider remains associated with devices and others models.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

Multiple Providers can be associated with the subsystem., where the Provider is either *primary* or *ancillary*. Entities associated with the service are owned by the former provider, where the latter makes use of this information (priority is given to the main Provider). The device subsystem is one of example of services that supports multiple providers.

In the core the Manager component lies and its job is to receive information from Provider and to provide this information further to applications and other services. Its interfaces include [19]: a northbound `Service` interface, an `AdminService` interface, a southbound `ProviderRegistry` interface, a southbound `ProviderService` interface.

Store's specific implementation is highly related to the Manager in the Core, as the information received by the latter needs to be indexed, persisted and synchronized with the former, where robustness and consistency of information must be ensured across multiple ONOS instances.

The application consumes and treats the information gained by the Manager through the `AdminService` and `Service` interfaces and it has a wide range of functions, including displaying the network topology in the web browser and setting the path for network traffic [19].

Like Provider, each application is associated with a unique `ApplicationId`, which is used by ONOS to track context associated with this specific application.

Events and descriptions are the two basic information units distributed in ONOS, which are unaltered once created and they are connected with specific network elements and concepts. More specifically, descriptions are used to pass information about elements on the southbound API, whereas events are used by Managers to notify about changes, and by Stores to notify about events in a distributed setting.

Based on input from the Manager, Events are generated by the Store, and once created they are dispatched to interested listeners via the `StoreDelegate` interface, which, in turn, invokes the `EventDeliveryService`. More specifically, the `StoreDelegate` guides the event out of the store, and the `EventDeliveryService` makes sure that the event only reaches interested listeners. These two components are in the Manager, which provides the implementation class of the `StoreDelegate` to the Store.

Any components that implement the `EventListener` interface are referred to as Event listeners. Note that `EventListener` child interfaces are classified based on the type of `Event` subclass.

Figure 17 explains the relationship of these components. It is noted that model objects are an ONOS protocol-independent way to represent various network elements and attributes.

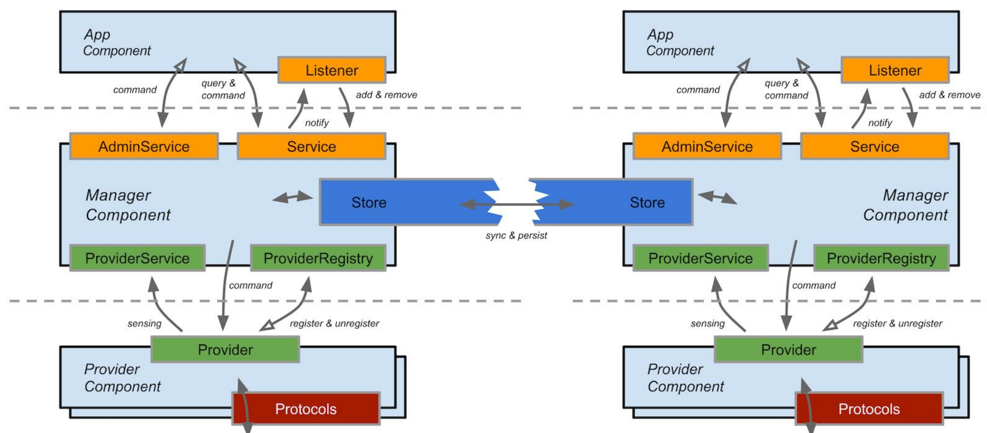


Figure 16: Subsystem structure [19]

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

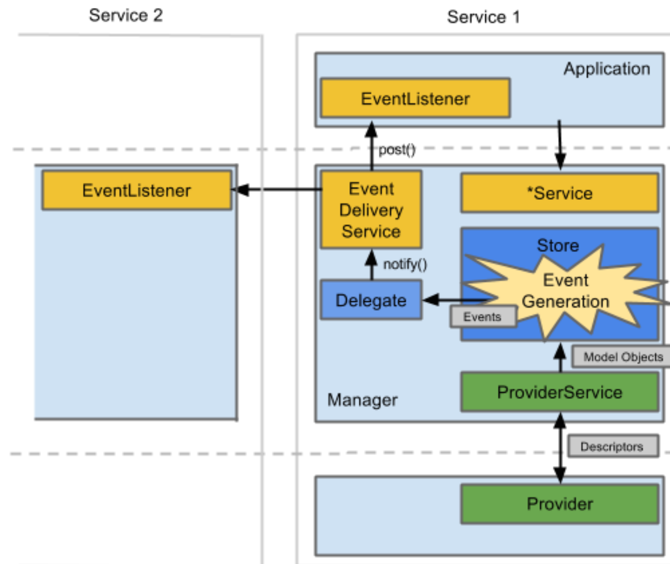


Figure 17: Events

5.5 Network State Construction

Control plane holds many key piece of information, such as the network state [20]. Information is gathered by the control plane, and then is provided to the applications. Actions need to be done in order to maintain extensibility and component reuse.

Network discovery and configuration are used as mechanisms in order to build ONOS's protocol-agnostic topology, combining both the advantages of these mechanisms. Furthermore, ONOS maintains protocol-agnostic and protocol-specific network element and state representations that can be translated from one to the other [20]. On top of that, rich data types are used instead of Java primitives for clarity reasons, when this is possible.

It is intresting to pay attention to the network topology (Device, Host, Link, Path, and others), which is described in [20]

Speaking of application level, directives for the network are expressed as high-level flow rules given as Criteria (Match) and Treatment (Action) pairs [20].

Packets, such as from network traffic, and those to be injected into the network, have analogues to the OpenFlow PacketIn and PacketOut [20].

- *OutboundPacket* - Protocol agnostic representation of a synthetic packet to be emitted on the network. This includes information about where to emit this packet.
- *InboundPacket* - Protocol agnostic representation of a packet sent to the controller by a device. This enables reactive packet processing by making PacketIns available to providers and applications to use as needed, e.g. host tracking, link detection.

There is a dependency amongst objects [20]; for example, Ports cannot exist without a Device or Links cannot exist without Ports, which serve as endpoints to the former. Therefore it is considered Devices to be a first-class entity in ONOS's network representations.

ONOS uses the following terminology [20], and based on this terminology an overview of network configuration is given in Figure 18:

- a *subject* is referred to an object to be configured via this subsystem. For instance: a `DeviceId` for a network device
- a *config* is a set of exposed tunables for a given object. For instance: a `BasicDeviceConfig` allows a device's type and southbound driver to be set or changed
- a *key*, or *subject key*, is a string name for a subject and it is used as the key for the JSON field containing the configuration values. For instance: Device configurations can be found in the field with key "devices".

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

- a *config key* is a string name for a configuration class and it is also used as a JSON field key. For instance: the key "basic" refers to the general configurations allowed for a device
- A *config operator* reconciles different sources of network configuration information for a given object

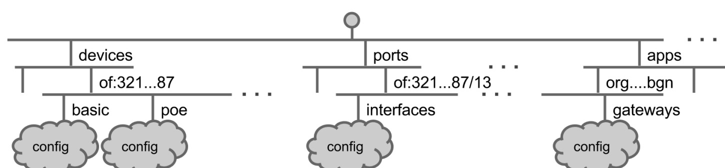


Figure 18: Overview of network configuration

The interested reader who wants to deep further into the Components is referred to [20].

5.6 Device Subsystem

The Device subsystem's job is to discover and track devices that comprise the network, enabling operators and applications to control them, and most of ONOS's core subsystems depend on the Device and Port model objects, which are created and managed by this subsystem [20].
The Device subsystem consists of [20]:

- a `DeviceManager`, which can interface with multiple `Providers` through a `DeviceProviderService` interface and multiple listeners through a `DeviceService` interface
- `DeviceProviders`, supported from their own network protocol libraries or means to interface with the network
- a `DeviceStore`, tracking Device model objects and generating `DeviceEvents`

`OpenFlowDeviceProvider`, is used by ONOS to interact with OpenFlow. The table below gives the major representations translated across the two tiers (note that various network components and properties are presented by ONOS as protocol-agnostic model objects at the core tier, while as protocol-specific objects at the provider tier) [20].

Device	OpenFlowSwitch
DeviceId/ElementId	Dpid
Port	OFPortDesc
MastershipRole	RoleState

The OpenFlow southbound consists of the `OpenFlowDeviceProvider` and as well as the OpenFlow driver components, which are referred as the OpenFlow subsystem. Using the Java protocol bindings generated with Loxi[41], the OpenFlow subsystem implements the controller-side behavior of the OpenFlow protocol [20].
Figure 19 illustrates the southbound.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

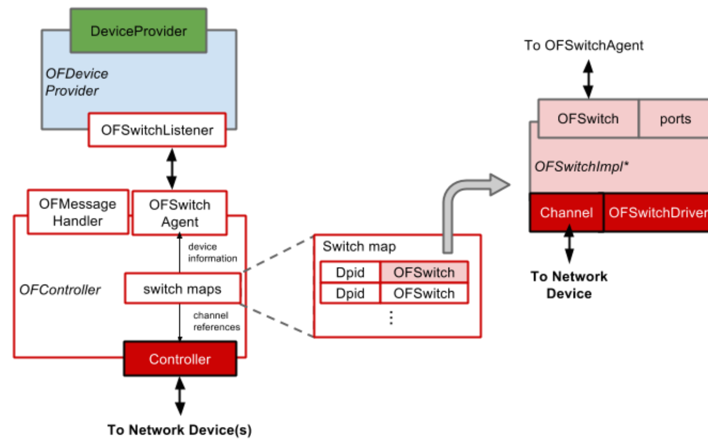


Figure 19: OF provider [20]

OpenFlowController is responsible for the coordination of the OpenFlow functions. It generates OpenFlow events that providers may subscribe to. Note that providers can be one or more of the following listeners [20]: OpenFlowSwitchListener, OpenFlowEventListener, PacketListener which are listeners for switch events, OpenFlow messages and incoming trafic packets, respectively.

OpenFlowController's job is also to establish and manage the communication channels for each of the Switch objects [20]. Controller establishes the connections, while the state of each connected switch is kept track of by the OpenFlowSwitchAgent.

The OpenFlow Switch object is the OpenFlow subsystem's representation of a network device and it is complete with a set of ports, device information, unique identifier, and a channel reference to the actual device connected on the other side [20].

A Switch object has two types of interfaces [20]: OpenFlowSwitch and OpenFlowSwitchDriver facing in specifing direction (north, towards Providers and south towards the channel and Controller, respectively)

5.7 Device Driver Subsystem

The primary purpose of this subsystem is to isolate device-specific code so it does not spread-out throughout the rest of the system [20].

A Driver is a representation of a specific family of devices or a specific device and it holds the following properties [20]: it has a unique name, it supports set of Behaviour classes, it may "inherit" behaviours from another Driver and it may be "abstract"

To implement aDriver interface, ONOS provides a class named DefaultDriver.

An entity DriverProvider can provide device drivers and their behaviors:

- Set<Driver> getDrivers()

The service DriverAdminService tracks and manages device drivers indirectly by managing driver providers as follows [20]:

- Set<DriverProvider> getProviders()
- registerProvider(DriverProvider)
- unregisterProvider(DriverProvider)

DriverService is the primary service that applications and other ONOS subsystems can use in order to locate appropriate drivers for the device through the following [20]:

- by driver name
- by device manufacturer, H/W version & S/W version
- by supported Behaviour
- by device ID

DriverData is a container for data learned about a device in prior interactions. It provides Behaviours for talking about a device and has parent Driver [20].

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

6 Next Generation Software Defined Networking

6.1 Stratum

Stratum[7] is defined as "an open source silicon-independent switch operating system for software defined networks". It builds an open and minimal production-ready distribution for white box switches. A set of next-generation SDN interfaces are exposed by Stratum, including P4Runtime and OpenConfig. As a results, interchangeability of forwarding devices and programmability of forwarding behaviors are enabled.

Furthermore, broadens the scope of SDN to include full lifecycle control, configuration, and operations interfaces. Stratum represents a promising software component of SDN solutions, while it implements the latest SDN/centric northbound interfaces, such as P4, P4Runtime, gNMI/OpenConfig, and gNOI. It does not include control protocols; rather it is designed in that way in order to support either an external Network OS or to work with NOS functions running on the same embedded switch. Stratum project is depicted in Figure 20.

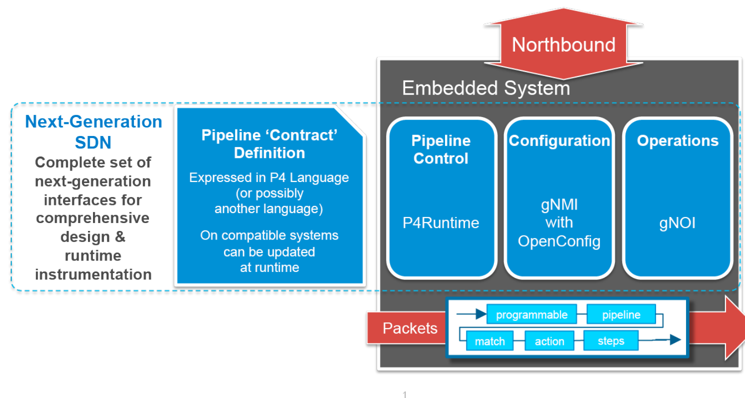


Figure 20: Stratum Project

Amongst the others, Stratum avoids the vendor lock-in of today's data planes or proprietary silicon interfaces and importantly avoids closed software APIs. In this way, easy integration of devices into operator networks is available. Stratum delivers a complete white box switch solution to realize the software defined promise of SDN [21].

Stratum offers unique advantages when it comes to work with modern SDN controllers. The unique level of programmability fits well with modern switches/siliconsfor controllerslike ONOS. On top of that, Stratum is a versatile platform leveraging the capabilities of other open source projects, such as P4 and ONIE [46]. Last but not least, Stratum provides an open source solution for fully-programmable data planes.

Figure 21 depicts a Stratum Controller.

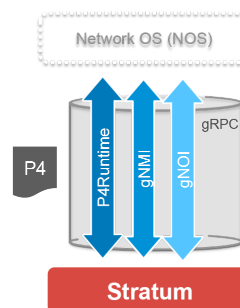


Figure 21: Stratum Controller

Stratum comes with a docker [39] image that can execute a mininet emulated network using stratum_bmv2 as the default switch. The network topology shown in Figure 22, was created using the aforementioned image to test and showcase P4 runtime, gNMI [47], ONOS as control plane, an implementation of IPv6 routing with ECMP [29] and an implementation of SRv6 [27] .

[7] https://opennetworking.org/stratum/

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

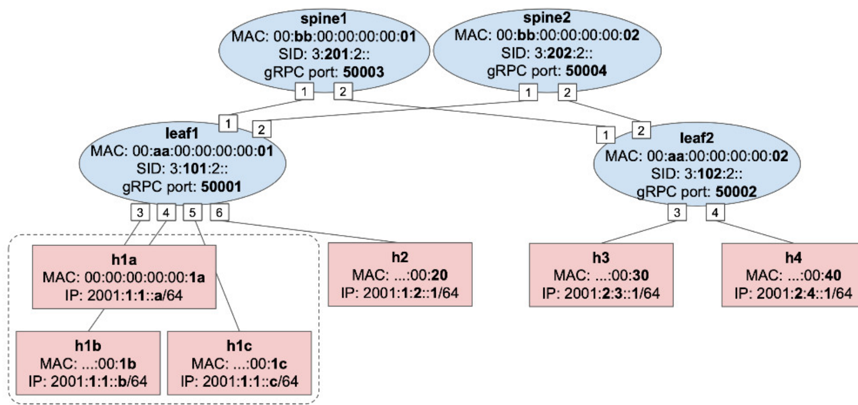


Figure 22: Mininet Topology

The topology includes four switches, arranged in a 2x2 fabric manner. Six hosts attached to leaf switches are included, as well. Three hosts h1a, h1b, and h1c, are part of the same IPv6 subnet. Each host is configured with IPv6 gateway address on the same subnet as the host, but ending with ....::ff, e.g, for h1a the gateway address is 2001:1:1::ff.

What follows next is an introduction to the P4Runtime API. The main.p4 file is used in all of the showcases presented in this thesis. For some showcases where extra code is required, it will be presented as well.

main.p4

```
#include <core.p4>
#include <v1model.p4>

// CPU_PORT specifies the P4 port number associated to controller packet-in and
// packet-out. All packets forwarded via this port will be delivered to the
// controller as P4Runtime PacketIn messages. Similarly, PacketOut messages from
// the controller will be seen by the P4 pipeline as coming from the CPU_PORT.
#define CPU_PORT 255

// CPU_CLONE_SESSION_ID specifies the mirroring session for packets to be cloned
// to the CPU port. Packets associated with this session ID will be cloned to
// the CPU_PORT as well as being transmitted via their egress port (set by the
// bridging/routing/acl table). For cloning to work, the P4Runtime controller
// needs first to insert a CloneSessionEntry that maps this session ID to the
// CPU_PORT.
#define CPU_CLONE_SESSION_ID 99

typedef bit<9>   port_num_t;
typedef bit<48>  mac_addr_t;
typedef bit<16>  mcast_group_id_t;
typedef bit<32>  ipv4_addr_t;
typedef bit<128> ipv6_addr_t;
typedef bit<16>  l4_port_t;

const bit<16> ETHERTYPE_IPV4 = 0x0800;
const bit<16> ETHERTYPE_IPV6 = 0x86dd;

const bit<8> IP_PROTO_ICMP   = 1;
const bit<8> IP_PROTO_TCP    = 6;
const bit<8> IP_PROTO_UDP    = 17;
const bit<8> IP_PROTO_ICMPV6 = 58;

const mac_addr_t IPV6_MCAST_01 = 0x33_33_00_00_00_01;

const bit<8> ICMP6_TYPE_NS = 135;
const bit<8> ICMP6_TYPE_NA = 136;

const bit<8> NDP_OPT_TARGET_LL_ADDR = 2;

const bit<32> NDP_FLAG_ROUTER    = 0x80000000;
const bit<32> NDP_FLAG_SOLICITED = 0x40000000;
const bit<32> NDP_FLAG_OVERRIDE  = 0x20000000;


```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
//------------------------------------------------------------------------------
// HEADER DEFINITIONS
//------------------------------------------------------------------------------

header ethernet_t {
    mac_addr_t  dst_addr;
    mac_addr_t  src_addr;
    bit<16>     ether_type;
}

header ipv4_t {
    bit<4>   version;
    bit<4>   ihl;
    bit<6>   dscp;
    bit<2>   ecn;
    bit<16>  total_len;
    bit<16>  identification;
    bit<3>   flags;
    bit<13>  frag_offset;
    bit<8>   ttl;
    bit<8>   protocol;
    bit<16>  hdr_checksum;
    bit<32>  src_addr;
    bit<32>  dst_addr;
}

header ipv6_t {
    bit<4>    version;
    bit<8>    traffic_class;
    bit<20>   flow_label;
    bit<16>   payload_len;
    bit<8>    next_hdr;
    bit<8>    hop_limit;
    bit<128>  src_addr;
    bit<128>  dst_addr;
}

header tcp_t {
    bit<16>  src_port;
    bit<16>  dst_port;
    bit<32>  seq_no;
    bit<32>  ack_no;
    bit<4>   data_offset;
    bit<3>   res;
    bit<3>   ecn;
    bit<6>   ctrl;
    bit<16>  window;
    bit<16>  checksum;
    bit<16>  urgent_ptr;
}

header udp_t {
    bit<16> src_port;
    bit<16> dst_port;
    bit<16> len;
    bit<16> checksum;
}

header icmp_t {
    bit<8>   type;
    bit<8>   icmp_code;
    bit<16>  checksum;
    bit<16>  identifier;
    bit<16>  sequence_number;
    bit<64>  timestamp;
}

header icmpv6_t {
    bit<8>   type;
    bit<8>   code;
    bit<16>  checksum;
}

header ndp_t {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
    bit<32>      flags;
    ipv6_addr_t  target_ipv6_addr;
    // NDP option.
    bit<8>       type;
    bit<8>       length;
    bit<48>      target_mac_addr;
}

// Packet-in header. Prepended to packets sent to the CPU_PORT and used by the
// P4Runtime server (Stratum) to populate the PacketIn message metadata fields.
// Here we use it to carry the original ingress port where the packet was
// received.
@controller_header("packet_in")
header cpu_in_header_t {
    port_num_t  ingress_port;
    bit<7>      _pad;
}

// Packet-out header. Prepended to packets received from the CPU_PORT. Fields of
// this header are populated by the P4Runtime server based on the P4Runtime
// PacketOut metadata fields. Here we use it to inform the P4 pipeline on which
// port this packet-out should be transmitted.
@controller_header("packet_out")
header cpu_out_header_t {
    port_num_t  egress_port;
    bit<7>      _pad;
}

struct parsed_headers_t {
    cpu_out_header_t cpu_out;
    cpu_in_header_t cpu_in;
    ethernet_t ethernet;
    ipv4_t ipv4;
    ipv6_t ipv6;
    srv6h_t srv6h;
    srv6_list_t[SRV6_MAX_HOPS] srv6_list;
    tcp_t tcp;
    udp_t udp;
    icmp_t icmp;
    icmpv6_t icmpv6;
    ndp_t ndp;
}

struct local_metadata_t {
    l4_port_t   l4_src_port;
    l4_port_t   l4_dst_port;
    bool        is_multicast;
    ipv6_addr_t next_srv6_sid;
    bit<8>      ip_proto;
    bit<8>      icmp_type;
}


//------------------------------------------------------------------------------
// INGRESS PIPELINE
//------------------------------------------------------------------------------

parser ParserImpl (packet_in packet,
                   out parsed_headers_t hdr,
                   inout local_metadata_t local_metadata,
                   inout standard_metadata_t standard_metadata)
{
    state start {
        transition select(standard_metadata.ingress_port) {
            CPU_PORT: parse_packet_out;
            default: parse_ethernet;
        }
    }

    state parse_packet_out {
        packet.extract(hdr.cpu_out);
        transition parse_ethernet;
    }

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.ether_type){
            ETHERTYPE_IPV4: parse_ipv4;
            ETHERTYPE_IPV6: parse_ipv6;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        local_metadata.ip_proto = hdr.ipv4.protocol;
        transition select(hdr.ipv4.protocol) {
            IP_PROTO_TCP: parse_tcp;
            IP_PROTO_UDP: parse_udp;
            IP_PROTO_ICMP: parse_icmp;
            default: accept;
        }
    }

    state parse_ipv6 {
        packet.extract(hdr.ipv6);
        local_metadata.ip_proto = hdr.ipv6.next_hdr;
        transition select(hdr.ipv6.next_hdr) {
            IP_PROTO_TCP: parse_tcp;
            IP_PROTO_UDP: parse_udp;
            IP_PROTO_ICMPV6: parse_icmpv6;
            IP_PROTO_SRV6: parse_srv6;
            default: accept;
        }
    }

    state parse_tcp {
        packet.extract(hdr.tcp);
        local_metadata.l4_src_port = hdr.tcp.src_port;
        local_metadata.l4_dst_port = hdr.tcp.dst_port;
        transition accept;
    }

    state parse_udp {
        packet.extract(hdr.udp);
        local_metadata.l4_src_port = hdr.udp.src_port;
        local_metadata.l4_dst_port = hdr.udp.dst_port;
        transition accept;
    }

    state parse_icmp {
        packet.extract(hdr.icmp);
        local_metadata.icmp_type = hdr.icmp.type;
        transition accept;
    }

    state parse_icmpv6 {
        packet.extract(hdr.icmpv6);
        local_metadata.icmp_type = hdr.icmpv6.type;
        transition select(hdr.icmpv6.type) {
            ICMP6_TYPE_NS: parse_ndp;
            ICMP6_TYPE_NA: parse_ndp;
            default: accept;
        }
    }

    state parse_ndp {
        packet.extract(hdr.ndp);
        transition accept;
    }
}


control VerifyChecksumImpl(inout parsed_headers_t hdr,
                           inout local_metadata_t meta)
{
    // Not used here. We assume all packets have valid checksum, if not, we let
    // the end hosts detect errors.
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
267     apply { /* EMPTY */ }
268 }
269
270
271 control IngressPipeImpl (inout parsed_headers_t    hdr,
272                          inout local_metadata_t    local_metadata,
273                          inout standard_metadata_t standard_metadata) {
274
275     // Drop action shared by many tables.
276     action drop() {
277         mark_to_drop(standard_metadata);
278     }
279
280     action set_egress_port(port_num_t port_num) {
281         standard_metadata.egress_spec = port_num;
282     }
283
284     table l2_exact_table {
285         key = {
286             hdr.ethernet.dst_addr: exact;
287         }
288         actions = {
289             set_egress_port;
290             @defaultonly drop;
291         }
292         const default_action = drop;
293         // The @name annotation is used here to provide a name to this table
294         // counter, as it will be needed by the compiler to generate the
295         // corresponding P4Info entity.
296         @name("l2_exact_table_counter")
297         counters = direct_counter(CounterType.packets_and_bytes);
298     }
299
300     // --- l2_ternary_table (for broadcast/multicast entries) ------------------
301
302     action set_multicast_group(mcast_group_id_t gid) {
303         // gid will be used by the Packet Replication Engine (PRE) in the
304         // Traffic Manager--located right after the ingress pipeline, to
305         // replicate a packet to multiple egress ports, specified by the control
306         // plane by means of P4Runtime MulticastGroupEntry messages.
307         standard_metadata.mcast_grp = gid;
308         local_metadata.is_multicast = true;
309     }
310
311     table l2_ternary_table {
312         key = {
313             hdr.ethernet.dst_addr: ternary;
314         }
315         actions = {
316             set_multicast_group;
317             @defaultonly drop;
318         }
319         const default_action = drop;
320         @name("l2_ternary_table_counter")
321         counters = direct_counter(CounterType.packets_and_bytes);
322     }
323
324     // *** ACL
325     //
326     // Provides ways to override a previous forwarding decision, for example
327     // requiring that a packet is cloned/sent to the CPU, or dropped.
328     //
329     // We use this table to clone all NDP packets to the control plane, so to
330     // enable host discovery. When the location of a new host is discovered, the
331     // controller is expected to update the L2 and L3 tables with the
332     // corresponding bridging and routing entries.
333
334     action send_to_cpu() {
335         standard_metadata.egress_spec = CPU_PORT;
336     }
337
338     action clone_to_cpu() {
339         // Cloning is achieved by using a v1model-specific primitive. Here we
340         // set the type of clone operation (ingress-to-egress pipeline), the
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
341         // clone session ID (the CPU one), and the metadata fields we want to
342         // preserve for the cloned packet replica.
343         clone3(CloneType.I2E, CPU_CLONE_SESSION_ID, { standard_metadata.ingress_port });
344     }
345 
346     table acl_table {
347         key = {
348             standard_metadata.ingress_port: ternary;
349             hdr.ethernet.dst_addr:          ternary;
350             hdr.ethernet.src_addr:          ternary;
351             hdr.ethernet.ether_type:        ternary;
352             local_metadata.ip_proto:        ternary;
353             local_metadata.icmp_type:       ternary;
354             local_metadata.l4_src_port:     ternary;
355             local_metadata.l4_dst_port:     ternary;
356         }
357         actions = {
358             send_to_cpu;
359             clone_to_cpu;
360             drop;
361         }
362         @name("acl_table_counter")
363         counters = direct_counter(CounterType.packets_and_bytes);
364     }
365 
366     apply {
367         bool do_l3_l2 = true;
368 
369         if (do_l3_l2) {
370             // L2 bridging logic. Apply the exact table first...
371             if (!l2_exact_table.apply().hit) {
372                 // ...if an entry is NOT found, apply the ternary one in case
373                 // this is a multicast/broadcast NDP NS packet.
374                 l2_ternary_table.apply();
375             }
376         }
377 
378         // Lastly, apply the ACL table.
379         acl_table.apply();
380     }
381 }
382 
383 
384 control EgressPipeImpl (inout parsed_headers_t hdr,
385                         inout local_metadata_t local_metadata,
386                         inout standard_metadata_t standard_metadata) {
387     apply {
388         // If this is a multicast packet (flag set by l2_ternary_table), make
389         // sure we are not replicating the packet on the same port where it was
390         // received. This is useful to avoid broadcasting NDP requests on the
391         // ingress port.
392         if (local_metadata.is_multicast == true &&
393               standard_metadata.ingress_port == standard_metadata.egress_port) {
394             mark_to_drop(standard_metadata);
395         }
396     }
397 }
398 
399 
400 control ComputeChecksumImpl(inout parsed_headers_t hdr,
401                             inout local_metadata_t local_metadata)
402 {
403     apply {
404         // The following is used to update the ICMPv6 checksum of NDP
405         // NA packets generated by the ndp reply table in the ingress pipeline.
406         // This function is executed only if the NDP header is present.
407         update_checksum(hdr.ndp.isValid(),
408             {
409                 hdr.ipv6.src_addr,
410                 hdr.ipv6.dst_addr,
411                 hdr.ipv6.payload_len,
412                 8w0,
413                 hdr.ipv6.next_hdr,
414                 hdr.icmpv6.type,
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
415                  hdr.icmpv6.code,
416                  hdr.ndp.flags,
417                  hdr.ndp.target_ipv6_addr,
418                  hdr.ndp.type,
419                  hdr.ndp.length,
420                  hdr.ndp.target_mac_addr
421              },
422              hdr.icmpv6.checksum,
423              HashAlgorithm.csum16
424          );
425      }
426  }
427
428
429  control DeparserImpl(packet_out packet, in parsed_headers_t hdr) {
430      apply {
431          packet.emit(hdr.cpu_in);
432          packet.emit(hdr.ethernet);
433          packet.emit(hdr.ipv4);
434          packet.emit(hdr.ipv6);
435          packet.emit(hdr.tcp);
436          packet.emit(hdr.udp);
437          packet.emit(hdr.icmp);
438          packet.emit(hdr.icmpv6);
439          packet.emit(hdr.ndp);
440      }
441  }
442
443  V1Switch(
444      ParserImpl(),
445      VerifyChecksumImpl(),
446      IngressPipeImpl(),
447      EgressPipeImpl(),
448      ComputeChecksumImpl(),
449      DeparserImpl()
450  ) main;
```

To compile the P4 program for the bmv2 simple_switch target, the open source P4 compiler p4c is used, which includes a backend for this specific target named p4c-bm2-ss. The OpenNetworking provides a docker image for p4c and is used to compile the main.p4 file. The command used to compile the main.p4 file is shown in the following snippet:

```
1  $ docker run --rm -v /workdir -w /workdir opennetworking/p4c:stable \
2                  p4c-bm2-ss --arch v1model -o p4src/build/bmv2.json \
3                  --p4runtime-files p4src/build/p4info.txt --Wdisable=unsupported \
4                  p4src/main.p4
```

Using these arguments when calling p4c-bm2-ss, the compiler is instructed to:

- Compile for the v1model architecture (–arch argument)
- Put the main output in p4src/build/bmv2.json (-o)
- Generate a P4Info file in p4src/build/p4info.txt (–p4runtime-files)
- Ignore some warnings about unsupported features (–Wdisable=unsupported), it is ok to ignore such warnings here, as they are generated because of a bug in p4c

The compiler outputs two files namely bmv2.json and p4info.txt. The bmv2.json file defines a configuration for the bmv2 simple_switch target in JSON format. When simple_switch receives a new packet, it uses this configuration to process the packet in a way that is consistent with the P4 program. The p4info.txt file contains an instance of a P4Info schema for our P4 program, expressed using the protobuf Text format.

When the mininet container starts, a set of files related to the execution of each stratum_bmv2 instance is generated. Examples include:

- leaf1/stratum_bmv2.log: contains the stratum_bmv2 log for switch leaf1
- leaf1/chassis-config.txt: the stratum "chassis config" file used to specify the initial port configuration to use at switch startup
- leaf1/write-reqs.txt: a log of all P4Runtime write requests processed by the switch, the file might not exist if the switch has not received any write request

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

To program leaf1 using P4Runtime, the P4Runtime shell [36] is used, which is an interactive Python CLI that can be used to connect to a P4Runtime server and can run P4Runtime commands. It can be used to create, read, update, and delete flow table entries.

The shell can be started in two modes, with or without a P4 pipeline config. In the first case, the shell will take care of pushing the given pipeline config to the switch using the P4Runtime SetPipelineConfig RPC. In the second case, the shell will try to retrieve the P4Info that is currently configured in the switch.

In both cases, the shell makes use of the P4Info file to:

- allow specifying runtime entities such as table entries using P4Info names rather then numeric IDs, much easier to remember and read
- provide autocompletion
- validate the CLI commands

Finally, when connecting to a P4Runtime server, the specification mandates that a mastership election ID is provided to be able to write state, such as the pipeline config and table entries.

To connect the P4Runtime shell to leaf1 and push the pipeline configuration obtained from the compilation of main.p4 file, the following command is used:

```
$ docker run -ti p4lang/p4runtime-sh --grpc-addr localhost:50001 \
                                     --config p4src/build/p4info.txt,p4src/build/bmv2.json \
                                     --election-id 0,1
```

(as a note: –grpc-addr localhost:50001 is used because the mininet container is executed locally, and 50001 is the TCP port associated to the gRPC server exposed by leaf1)

If the shell is started successfully, the following output should be presented:

```
*** Connecting to P4Runtime server at host.docker.internal:50001 ...
*** Welcome to the IPython shell for P4Runtime ***
P4Runtime sh >>>
```

Some of the available commands are tables, actions, action_profiles, counters, direct_counters, and other named after the P4Info message fields, to query information about P4Info objects. Commands such as table_entry, action_profile_member, action_profile_group, counter_entry, direct_counter_entry, meter_entry, direct_meter_entry, multicast_group_entry, and clone_session_entry, can be used to read/write the corresponding P4Runtime entities. Additionaly, P4Runtime shell provides a help menu, typing the command name followed by ?, e.g. table_entry?. Also, the shell supports autocompletion when pressing tab.

In order to showcase how the P4 Runtime can control the forwarding plane of the switches, a bridging connectivity test is performed. To be able to ping two IPv6 hosts in the same subnet, first, the hosts need to resolve their respective MAC address using the Neighbor Discovery Protocol (NDP) [24]. This is equivalent to ARP in IPv4 networks. For example, when trying to ping h1b from h1a, h1a first an NDP Neighbor Solicitation (NS) message is generated to resolve the MAC address of h1b. Once h1b receives the NDP NS message, it should reply with an NDP Neighbor Advertisement (NA) with its own MAC address. Now both host are aware of each other MAC address and the ping packets can be exchanged.

In the main.p4 file is defined a way on how to handle NDP packets using P4Runtime (l2_ternary_table lines 311-322). However, for the sake of simplicity two static NDP entries is inserted to the hosts, to add an NDP entry to h1a, mapping h1b's IPv6 address (2001:1:1::B) to its MAC address (00:00:00:00:00:1B):

```
mininet> h1a ip -6 neigh replace 2001:1:1::B lladdr 00:00:00:00:00:1B dev h1a-eth0
```

and vice versa, to add an NDP entry to h1b to resolve h1a's address:

```
mininet> h1b ip -6 neigh replace 2001:1:1::A lladdr 00:00:00:00:00:1A dev h1b-eth0
```

A ping between h1a and h1b should not work, yet, because P4Runtime table entry for forward these packets is not inserted. To be able to forward ping packets, two table entries are needed to be added on l2_exact_table in leaf1, one that matches on destination MAC address of h1b and forwards traffic to port 4, where h1b is attached, and one that matches on destination MAC address of h1b and forwards traffic to port 3, where h1a is attached.

For this purpose, the P4Runtime shell is used to create and insert such entries. To create a table entry object:

```
P4Runtime sh >>> te = table_entry["P4INFO-TABLE-NAME"](action = "<P4INFO-ACTION-NAME>")
```

and to specify a match field:

```
P4Runtime sh >>> te.match["P4INFO-MATCH-FIELD-NAME"] = ("VALUE")
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

VALUE can be a MAC address expressed in Colon-Hexadecimal notation (e.g., 00:11:22:AA:BB:CC), or IP address in dot notation, or an arbitrary string. Based on the information contained in the P4Info, P4Runtime shell will internally convert that value to a protobuf byte string. To specify the values for the table entry action parameters:

```
P4Runtime sh >>> te.action["P4INFO-ACTION-PARAM-NAME"] = ("VALUE")
```

To show the table entry object in protobuf Text format:

```
P4Runtime sh >>> print(te)
```

The shell internally takes care of populating the fields of the corresponding protobuf message by using the content of the P4Info file. To insert the entry, this will issue a P4Runtime Write RPC to the switch:

```
P4Runtime sh >>> te.insert()
```

To read table entries from the switch, this will issue a P4Runtime Read RPC:

```
P4Runtime sh >>> for te in table_entry["P4INFO-TABLE-NAME"].read():
            ...:     print(te)
            ...:
```

After inserting the two entries, ping should work.

6.2 YANG

YANG[8] (Yet Another Next Generation) is "a data modelling language, providing a standardized way to model the operational and configuration data of a network device. YANG, being a language is being protocol independent, can then be converted into any encoding format, e.g. XML or JSON". From a YANG model, two things are important:

- the data tree organization from which we get the paths and leaf data types
- the semantics of the leaf nodes from the description field

A YANG model consists of various components, as shown in Figure 23, that will be discussed and presented with code snippets.

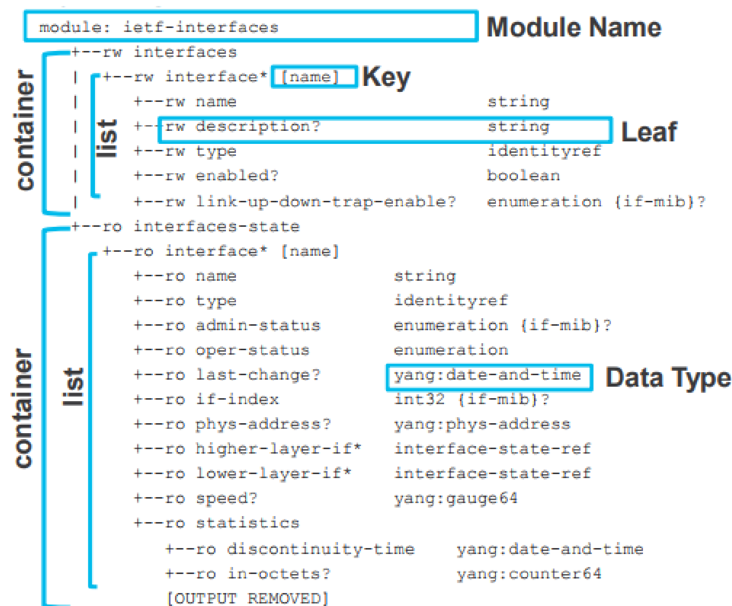


Figure 23: YANG Structure

A **module** is a self-contained tree of nodes. Modules are the smallest unit that can be "compiled" by YANG tools. A module contains:

[8]https://www.fir3net.com/Networking/Protocols/an-introduction-to-netconf-yang.html

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

- boilerplate, like a namespace, prefix for reference in other modules, description, version / revision history, etc.
- identities and derived types
- modular groupings
- a top-level container that defines tree of data nodes

An example snippet is shown below.

```
// A module is a self-contained tree of nodes
module demo-port {

    // YANG Boilerplate
    yang-version "1";
    namespace "https://opennetworking.org/yang/demo";
    prefix "demo-port";
    description "Demo model for managing ports";
    revision "2019-09-10" {
        description "Initial version";
        reference "1.0.0";
    }

    // ... insert rest of model here ...
}
```

YANG defines several built-in types including binary, bits, boolean, decimal64, empty, enumeration, identityref, int8, int16, int32, int64, string, uint8, uint16, uint32, uint64, decimal64. For the complete list, check out RFC 6020 [30] for YANG 1.0 or RFC 7950 [28] for YANG 1.1.

An **identity** is a globally unique, abstract, and untyped. Identities are used to identify something with explicit semantics and can be hierarchical. Derived types enable constraint of build-in types or other derived types, and they are defined using **typedef**. An example is presented below.

```
// Identities and Typedefs
identity SPEED {
    description "base type for port speeds";
}

identity SPEED_10GB {
    base SPEED;
    description "10 Gbps port speed";
}

typedef port-number {
    type uint16 {
        range 1..32;
    }
    description "New type for port number that ensure
      the number is between 1 and 32, inclusive";
}
```

A **grouping** is a reusable set of nodes (containers and leaves) that can be included in a **container**. However, on its own a grouping does not add any nodes to the module in which it is defined or imported. A **leaf** is a node that contains a value (built-in type or derived type) and has no children.

```
// Reusable groupings for port config and state
grouping port-config {
    description "Set of configurable attributes / leaves";
    leaf speed {
        type identityref {
            base demo-port:SPEED;
        }
        description "Configurable speed of a switch port";
    }
}

grouping port-state {
    description "Set of read-only state";
    leaf status {
        type boolean;
        description "Number";
    }
}
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

A **container** is a node with a set of children. Each module has one top-level or root container. A **list** is a node that contains a set of multiple children of the same type. Lists elements are identified by a **key**. Containers marked **config false** are state data that is read-only from a clients perspective. Typically, it is used for status or statistics.

```
container ports {
    description "The root container for port configuration and state";
        list port {
            key "port-number";
            description "List of ports on a switch";

            leaf port-number {
                type port-number;
                description "Port number (maps to the front panel port of a switch);
                  also the key for the port list";
            }

            // each individual will have the elements defined in the grouping
            container config {
                description "Configuration data for a port";
                uses port-config; // reference to grouping above
            }
            container state {
                config false; // makes child nodes read-only
                description "Read-only state for a port";
                uses port-state; // reference to grouping above
            }
        }
}
```

Other interesting terminology:

- **import** is used to include types, groupings, or other models
- **augment** is used to add to a previously defined schema
- **deviation** is used to indicate a device does not implement part of a schema

6.3 OpenConfig

OpenConfig [22] is "an informal working group of network operators sharing the goal of moving computer networks toward a more dynamic, programmable infrastructure by adopting software-defined networking principles such as declarative configuration and model-driven management and operations".

The initial focus in OpenConfig is on compiling a consistent set of vendor-neutral data models, written in YANG, based on actual operational needs from use cases and requirements from multiple network operators.

Streaming telemetry is a new paradigm for network monitoring, where data are streamed continuously from devices with efficient and incremental updates; operators can subscribe to the specific data items they need, using OpenConfig data models as the common interface [22].

OpenConfig focuses on compiling a consistent set of vendor neutral YANG data models continuously. These data models cover a large group of network elements, including routers and optical switches. Note that only a subset is relevant to the data plane. Figure 24 depicts OpenConfig Models. These are the models that Stratum is interested in: interfaces, lacp, platform, qos, vlan, system

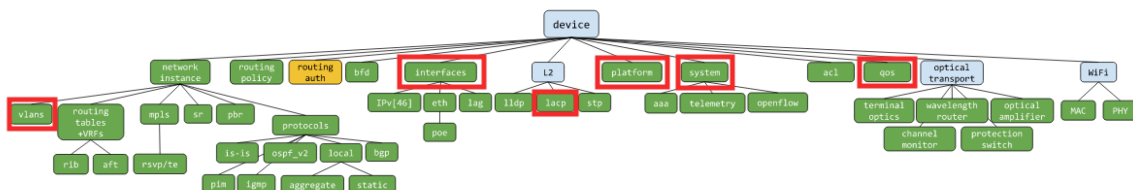


Figure 24: OpenConfig Models

There are several YANG-model agnostic protocols, figure 25, that can be used to get or set data that adheres to a model, like NETCONF [25], RESTCONF [26], and gNMI.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

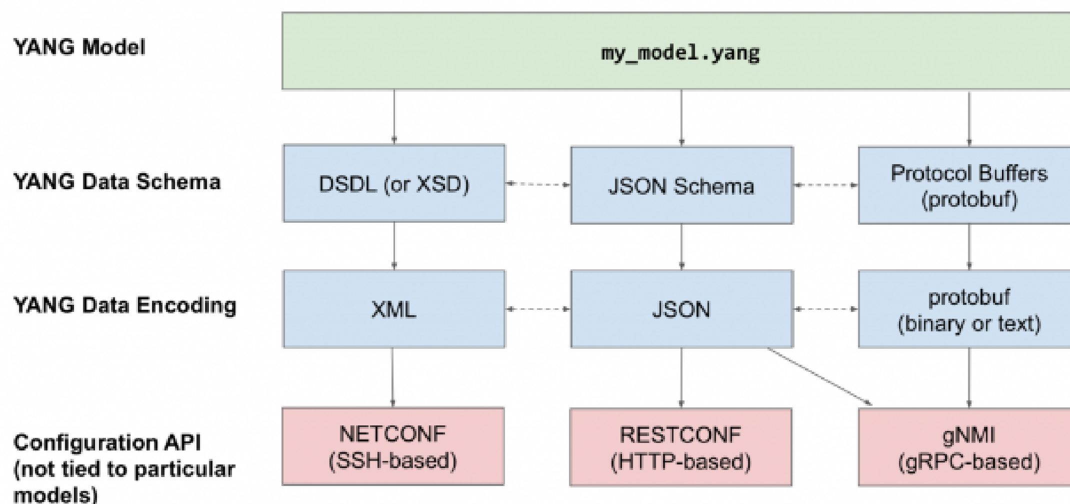


Figure 25: YANG Data Representation

6.3.1 gNMI

gNMI (gRPC Network Management Interface) is part of OpenConfig. It is a standardized protocol defined in protocol buffers to address YANG-based data models and to interact using a specific Network Management Interface. The main operations are: *SetRequest*, *SetResponse*, *GetRequest*, *GetResponse* and *Subscription* [11].

The purpose of the following showcase is to demonstrate YANG-enabled transport protocols, specifically gNMI. To do so, the gNMI client CLI [23] is used. To read all of the configuration from the Stratum switche leaf1 in our Mininet network:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 get /
```

the first part of the output shows the request that was made by the CLI:

```
REQUEST
path {
}
type: CONFIG
encoding: PROTO
```

the path being requested is the empty path, which means the root of the config tree, the type of data is just the config tree, and the requested encoding for the response is protobuf.

The second part of the output shows the response from Stratum:

```
RESPONSE
notification {
  update {
    path {
    }
    val {
      any_val {
        type_url: "type.googleapis.com/openconfig.Device"
        value: \252\221\231\304\001\... TRUNCATED
      }
    }
  }
}
```

the Stratum provides a response of type `openconfig.Device`, which is the top-level message defined in openconfig.proto. The response is the binary encoding of the data based on the protobuf message. The value is not human readable, but the reply can be translated using a utility that converts between the binary and textual representations of the protobuf message. The following command pipe the output through the converter utility then pipe that output to less to make scrolling easier:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 get / | \
$ docker run --rm -i bocon/yang-tools:latest oc-pb-decoder | \
less
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

The contents of the response should now be easier to read.

One of the benefits of gNMI is that it is "schema-less" encoding, which allows clients or devices to update only the paths that need to be updated. This is particularly useful for subscriptions. First, to try out the schema-less representation by requesting the configuration port between leaf1 and h1a:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 get \
    /interfaces/interface[name=leaf1-eth3]/config
```

this response containing 2 leafs under config, enabled and health-indicator:

```
RESPONSE
notification {
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "config"
      }
      elem {
        name: "enabled"
      }
    }
    val {
      bool_val: true
    }
  }
}
notification {
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "config"
      }
      elem {
        name: "health-indicator"
      }
    }
    val {
      string_val: "GOOD"
    }
  }
}
```

The schema-less representation provides and update for each leaf containing both the path the value of the leaf. The confirmation that the interface is enabled is that is set to true.

The next snippet shows how to subscribe to the ingress unicast packet counters for the interface on leaf1 attached to h1a, port 3:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 \
    --interval 1000 sub-sample \
    /interfaces/interface[name=leaf1-eth3]/state/counters/in-unicast-pkts
```

The first part of the output shows the request being made by the CLI. The subscription path, the type of subscription (sampling) and the sampling rate (every 1000ms, or 1s) are shown below.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
REQUEST
subscribe {
  subscription {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "counters"
      }
      elem {
        name: "in-unicast-pkts"
      }
    }
    mode: SAMPLE
    sample_interval: 1000
  }
  updates_only: true
}
```

The second part of the output is a stream of responses:

```
RESPONSE
update {
  timestamp: 1567895852136043891
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "counters"
      }
      elem {
        name: "in-unicast-pkts"
      }
    }
    val {
      uint_val: 1592
    }
  }
}
```

Each response has a timestamp, path, and new value. Because we are sampling, a new update should printed every second.

From mininet some traffic can be generated by a ping between hosts h1a and h1b. The uint_val should increase by 1 every second while ping is still running. If it is not exactly 1, then there could be other traffic like NDP messages contributing to the increase. To stop the gNMI subscription *Ctrl-C* is used.

Finally, to monitor link events using gNMI's on-change subscriptions, a subscription for the operational status of the first switch's first port can be started like this:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 sub-onchange \
    /interfaces/interface[name=leaf1-eth3]/state/oper-status
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

The following response indicates that port 1 is *UP*:

```
RESPONSE
update {
  timestamp: 1567896668419430407
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      string_val: "UP"
    }
  }
}
```

If the interface on leaf1 connected to h1a taken down with this command:

```
mininet> sh ifconfig leaf1-eth3 down
```

The response will show that the interface on leaf1 connected to h1a is *DOWN*:

```
RESPONSE
update {
  timestamp: 1567896891549363399
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      string_val: "DOWN"
    }
  }
}
```

The gNMI can also be used to disable or enable an interface, like this:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 set \
    /interfaces/interface[name=leaf1-eth3]/config/enabled \
    --bool-val false
```

The following request indicating the new value for the enabled leaf:

```
REQUEST
update {
  path {
    elem {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
      name: "interfaces"
    }
    elem {
      name: "interface"
      key {
        key: "name"
        value: "leaf1-eth3"
      }
    }
    elem {
      name: "config"
    }
    elem {
      name: "enabled"
    }
  }
  val {
    bool_val: false
  }
}
```

While this response indicating that the operational status of leaf1-eth3 is *DOWN*:

```
RESPONSE
update {
  timestamp: 1567896891549363399
  update {
    path {
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "leaf1-eth3"
        }
      }
      elem {
        name: "state"
      }
      elem {
        name: "oper-status"
      }
    }
    val {
      string_val: "DOWN"
    }
  }
}
```

and as a consequence the ping has stopped working.

To re-nable the port:

```
$ docker run --rm -it --network host bocon/gnmi-cli:latest --grpc-addr localhost:50001 set \
    /interfaces/interface[name=leaf1-eth3]/config/enabled \
    --bool-val true
```

after the above command execution the interface is *UP*, and the ping should resume.

6.4 ONOS as the control plane

This subsection showcase how to integrate ONOS built-in services for link and host discovery with P4 program. Such built-in services are based on the ability of switches to send data plane packets to the controller (packet-in) and vice versa (packet-out).

To make this work with main.p4, simple changes are needed to the code. Additionaly, changes to the pipeconf Java implementation is also required to enable ONOS's built-in apps to use packet-in/out via P4Runtime.

It will be presented in two parts:

- Enable packet I/O and verify link discovery
- Host discovery & L2 bridging

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

6.4.1 Enable Packet I/O

Here is presented how the controller packet I/O works with P4Runtime. The main.p4 provides support for carrying arbitrary metadata in P4Runtime PacketIn and PacketOut messages. Two special headers are defined and annotated with the standard P4 annotation `@controller_header`, lines 127-145 in the main.p4 file.

These headers are used to carry the original switch ingress port of a packet-in, and to specify the intended output port for a packet-out.

When the P4Runtime agent in stratum receives a packet from the switch CPU port, it expects to find the `cpu_in_header_t` header as the first one in the frame. It looks at the `controller_packet_metadata` part of the P4Info file to determine the number of bits to strip at the beginning of the frame and to populate the corresponding metadata field of the PacketIn message, including the ingress port as in this case.

Similarly, when stratum receives a P4Runtime PacketOut message, it uses the values found in the PacketOut's metadata fields to serialize and prepend a `cpu_out_header_t` to the frame before feeding it to the pipeline parser.

The main.p4 code already provides support for the following capabilities:

- Parse the cpu_out header, if the ingress port is the CPU one
- Emit the cpu_in header as the first one in the deparser
- Provide an ACL table with ternary match fields and an action to send or clone packets to the CPU port, which is used to generate a packet-ins

In order to provide complete packet-in/out support, the following modifications are required:

1. In the `IngressPipeImpl` block the packet egress port is set to that found in the `cpu_out` header then the `cpu_out header` is removed and the pipeline is exited.

```
control IngressPipeImpl (inout parsed_headers_t    hdr,
                         inout local_metadata_t    local_metadata,
                         inout standard_metadata_t standard_metadata) {

    ...

     apply {

        if (hdr.cpu_out.isValid()) {
            standard_metadata.egress_spec = hdr.cpu_out.egress_port;
            hdr.cpu_out.setInvalid();
            exit;
        }

        bool do_l3_l2 = true;

        ...

    }

```

2. In the `EgressPipeImpl` logic is implemented to check if the packet is to be forwarded to the CPU port, e.g, if in ingress on the ACL table with action send/clone_to_cpu is matched.

```
control EgressPipeImpl (inout parsed_headers_t hdr,
                        inout local_metadata_t local_metadata,
                        inout standard_metadata_t standard_metadata) {
    apply {

        if (standard_metadata.egress_port == CPU_PORT) {

            hdr.cpu_in.setValid();
            hdr.cpu_in.ingress_port = standard_metadata.ingress_port;
            exit;
        }

        ...

    }

    ...

}
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

The built-in apps of ONOS for basic sevices that is used in this showcase are

- *gui2*: ONOS web user interface (available at http://<ONOS_IP>:8181/onos/ui)
- *drivers.bmv2*: BMv2/Stratum drivers based on P4Runtime, gNMI, and gNOI
- *lldpprovider*: LLDP-based link discovery application
- *hostprovider*: Host discovery application

For ONOS to be integrated with the P4 program presented, a development of a new app is required that implements the pipeconf. This new app includes three pipeconf-related files presented below:

- *PipeconfLoader.java*: A component that registers the pipeconf at app activation
- *InterpreterImpl.java*: An implementation of the PipelineInterpreter driver behavior
- *PipelinerImpl.java*: An implementation of the Pipeliner driver behavior

PipeconfLoader.java

```
1  package org.onosproject.ngsdn.tutorial.pipeconf;
2
3  import org.onosproject.net.behaviour.Pipeliner;
4  import org.onosproject.net.driver.DriverAdminService;
5  import org.onosproject.net.driver.DriverProvider;
6  import org.onosproject.net.pi.model.DefaultPiPipeconf;
7  import org.onosproject.net.pi.model.PiPipeconf;
8  import org.onosproject.net.pi.model.PiPipelineInterpreter;
9  import org.onosproject.net.pi.model.PiPipelineModel;
10 import org.onosproject.net.pi.service.PiPipeconfService;
11 import org.onosproject.p4runtime.model.P4InfoParser;
12 import org.onosproject.p4runtime.model.P4InfoParserException;
13 import org.osgi.service.component.annotations.Activate;
14 import org.osgi.service.component.annotations.Component;
15 import org.osgi.service.component.annotations.Deactivate;
16 import org.osgi.service.component.annotations.Reference;
17 import org.osgi.service.component.annotations.ReferenceCardinality;
18 import org.slf4j.Logger;
19 import org.slf4j.LoggerFactory;
20
21 import java.net.URL;
22 import java.util.List;
23 import java.util.stream.Collectors;
24
25 import static org.onosproject.net.pi.model.PiPipeconf.ExtensionType.BMV2_JSON;
26 import static org.onosproject.net.pi.model.PiPipeconf.ExtensionType.P4_INFO_TEXT;
27 import static org.onosproject.ngsdn.tutorial.AppConstants.PIPECONF_ID;
28
29 /**
30  * Component that builds and register the pipeconf at app activation.
31  */
32 @Component(immediate = true, service = PipeconfLoader.class)
33 public final class PipeconfLoader {
34
35     private final Logger log = LoggerFactory.getLogger(getClass());
36
37     private static final String P4INFO_PATH = "/p4info.txt";
38     private static final String BMV2_JSON_PATH = "/bmv2.json";
39
40
41     @Reference(cardinality = ReferenceCardinality.MANDATORY)
42     private PiPipeconfService pipeconfService;
43
44     @Reference(cardinality = ReferenceCardinality.MANDATORY)
45     private DriverAdminService driverAdminService;
46
47     @Activate
48     public void activate() {
49         // Registers the pipeconf at component activation.
50         if (pipeconfService.getPipeconf(PIPECONF_ID).isPresent()) {
51             // Remove first if already registered, to support reloading of the
52             // pipeconf during the tutorial.
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
            pipeconfService.unregister(PIPECONF_ID);
        }
        removePipeconfDrivers();
        try {
            pipeconfService.register(buildPipeconf());
        } catch (P4InfoParserException e) {
            log.error("Unable to register " + PIPECONF_ID, e);
        }
    }

    @Deactivate
    public void deactivate() {
        // Do nothing.
    }

    private PiPipeconf buildPipeconf() throws P4InfoParserException {

        final URL p4InfoUrl = PipeconfLoader.class.getResource(P4INFO_PATH);
        final URL bmv2JsonUrlUrl = PipeconfLoader.class.getResource(BMV2_JSON_PATH);
        final PiPipelineModel pipelineModel = P4InfoParser.parse(p4InfoUrl);

        return DefaultPiPipeconf.builder()
                .withId(PIPECONF_ID)
                .withPipelineModel(pipelineModel)
                .addBehaviour(PiPipelineInterpreter.class, InterpreterImpl.class)
                .addBehaviour(Pipeliner.class, PipelinerImpl.class)
                .addExtension(P4_INFO_TEXT, p4InfoUrl)
                .addExtension(BMV2_JSON, bmv2JsonUrlUrl)
                .build();
    }

    private void removePipeconfDrivers() {
        List<DriverProvider> driverProvidersToRemove = driverAdminService
                .getProviders().stream()
                .filter(p -> p.getDrivers().stream()
                        .anyMatch(d -> d.name().endsWith(PIPECONF_ID.id())))
                .collect(Collectors.toList());

        if (driverProvidersToRemove.isEmpty()) {
            return;
        }

        log.info("Found {} outdated drivers for pipeconf '{}', removing...",
                 driverProvidersToRemove.size(), PIPECONF_ID);

        driverProvidersToRemove.forEach(driverAdminService::unregisterProvider);
    }
}
```

InterpreterImpl.java

```java
package org.onosproject.ngsdn.tutorial.pipeconf;

import com.google.common.collect.ImmutableList;
import com.google.common.collect.ImmutableMap;
import org.onlab.packet.DeserializationException;
import org.onlab.packet.Ethernet;
import org.onlab.util.ImmutableByteSequence;
import org.onosproject.net.ConnectPoint;
import org.onosproject.net.DeviceId;
import org.onosproject.net.Port;
import org.onosproject.net.PortNumber;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.driver.AbstractHandlerBehaviour;
import org.onosproject.net.flow.TrafficTreatment;
import org.onosproject.net.flow.criteria.Criterion;
import org.onosproject.net.packet.DefaultInboundPacket;
import org.onosproject.net.packet.InboundPacket;
import org.onosproject.net.packet.OutboundPacket;
import org.onosproject.net.pi.model.PiMatchFieldId;
import org.onosproject.net.pi.model.PiPacketMetadataId;
import org.onosproject.net.pi.model.PiPipelineInterpreter;
import org.onosproject.net.pi.model.PiTableId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.net.pi.runtime.PiPacketMetadata;
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
25 import org.onosproject.net.pi.runtime.PiPacketOperation;
26
27 import java.nio.ByteBuffer;
28 import java.util.Collection;
29 import java.util.List;
30 import java.util.Map;
31 import java.util.Optional;
32
33 import static java.lang.String.format;
34 import static java.util.stream.Collectors.toList;
35 import static org.onlab.util.ImmutableByteSequence.copyFrom;
36 import static org.onosproject.net.PortNumber.CONTROLLER;
37 import static org.onosproject.net.PortNumber.FLOOD;
38 import static org.onosproject.net.flow.instructions.Instruction.Type.OUTPUT;
39 import static org.onosproject.net.flow.instructions.Instructions.OutputInstruction;
40 import static org.onosproject.net.pi.model.PiPacketOperationType.PACKET_OUT;
41 import static org.onosproject.ngsdn.tutorial.AppConstants.CPU_PORT_ID;
42
43
44 /**
45  * Interpreter implementation.
46  */
47 public class InterpreterImpl extends AbstractHandlerBehaviour
48         implements PiPipelineInterpreter {
49
50
51     // From v1model.p4
52     private static final int V1MODEL_PORT_BITWIDTH = 9;
53
54     // From P4Info.
55     private static final Map<Criterion.Type, String> CRITERION_MAP =
56             new ImmutableMap.Builder<Criterion.Type, String>()
57                     .put(Criterion.Type.IN_PORT, "standard_metadata.ingress_port")
58                     .put(Criterion.Type.ETH_DST, "hdr.ethernet.dst_addr")
59                     .put(Criterion.Type.ETH_SRC, "hdr.ethernet.src_addr")
60                     .put(Criterion.Type.ETH_TYPE, "hdr.ethernet.ether_type")
61                     .put(Criterion.Type.IPV6_DST, "hdr.ipv6.dst_addr")
62                     .put(Criterion.Type.IP_PROTO, "local_metadata.ip_proto")
63                     .put(Criterion.Type.ICMPV4_TYPE, "local_metadata.icmp_type")
64                     .put(Criterion.Type.ICMPV6_TYPE, "local_metadata.icmp_type")
65                     .build();
66
67     /**
68      * Returns a collection of PI packet operations populated with metadata
69      * specific for this pipeconf and equivalent to the given ONOS
70      * OutboundPacket instance.
71      *
72      * @param packet ONOS OutboundPacket
73      * @return collection of PI packet operations
74      * @throws PiInterpreterException if the packet treatments cannot be
75      *                                executed by this pipeline
76      */
77     @Override
78     public Collection<PiPacketOperation> mapOutboundPacket(OutboundPacket packet)
79             throws PiInterpreterException {
80         TrafficTreatment treatment = packet.treatment();
81
82         // Packet-out in main.p4 supports only setting the output port,
83         // i.e. we only understand OUTPUT instructions.
84         List<OutputInstruction> outInstructions = treatment
85                 .allInstructions()
86                 .stream()
87                 .filter(i -> i.type().equals(OUTPUT))
88                 .map(i -> (OutputInstruction) i)
89                 .collect(toList());
90
91         if (treatment.allInstructions().size() != outInstructions.size()) {
92             // There are other instructions that are not of type OUTPUT.
93             throw new PiInterpreterException("Treatment not supported: " + treatment);
94         }
95
96         ImmutableList.Builder<PiPacketOperation> builder = ImmutableList.builder();
97         for (OutputInstruction outInst : outInstructions) {
98             if (outInst.port().isLogical() && !outInst.port().equals(FLOOD)) {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
                throw new PiInterpreterException(format(
                        "Packet-out on logical port '%s' not supported",
                        outInst.port()));
            } else if (outInst.port().equals(FLOOD)) {
                // To emulate flooding, we create a packet-out operation for
                // each switch port.
                final DeviceService deviceService = handler().get(DeviceService.class);
                for (Port port : deviceService.getPorts(packet.sendThrough())) {
                    builder.add(buildPacketOut(packet.data(), port.number().toLong()));
                }
            } else {
                // Create only one packet-out for the given OUTPUT instruction.
                builder.add(buildPacketOut(packet.data(), outInst.port().toLong()));
            }
        }
        return builder.build();
    }

    /**
     * Builds a pipeconf-specific packet-out instance with the given payload and
     * egress port.
     *
     * @param pktData    packet payload
     * @param portNumber egress port
     * @return packet-out
     * @throws PiInterpreterException if packet-out cannot be built
     */
    private PiPacketOperation buildPacketOut(ByteBuffer pktData, long portNumber)
            throws PiInterpreterException {

        // Make sure port number can fit in v1model port metadata bitwidth.
        final ImmutableByteSequence portBytes;
        try {
            portBytes = copyFrom(portNumber).fit(V1MODEL_PORT_BITWIDTH);
        } catch (ImmutableByteSequence.ByteSequenceTrimException e) {
            throw new PiInterpreterException(format(
                    "Port number %d too big, %s", portNumber, e.getMessage()));
        }

        // Create metadata instance for egress port.

        final String outPortMetadataName = "egress_port";

        final PiPacketMetadata outPortMetadata = PiPacketMetadata.builder()
                .withId(PiPacketMetadataId.of(outPortMetadataName))
                .withValue(portBytes)
                .build();

        // Build packet out.
        return PiPacketOperation.builder()
                .withType(PACKET_OUT)
                .withData(copyFrom(pktData))
                .withMetadata(outPortMetadata)
                .build();
    }

    /**
     * Returns an ONS InboundPacket equivalent to the given pipeconf-specific
     * packet-in operation.
     *
     * @param packetIn packet operation
     * @param deviceId ID of the device that originated the packet-in
     * @return inbound packet
     * @throws PiInterpreterException if the packet operation cannot be mapped
     *                                to an inbound packet
     */
    @Override
    public InboundPacket mapInboundPacket(PiPacketOperation packetIn, DeviceId deviceId)
            throws PiInterpreterException {

        // Find the ingress_port metadata.

        final String inportMetadataName = "ingress_port";

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
173         Optional<PiPacketMetadata> inportMetadata = packetIn.metadatas()
174                 .stream()
175                 .filter(meta -> meta.id().id().equals(inportMetadataName))
176                 .findFirst();
177 
178         if (!inportMetadata.isPresent()) {
179             throw new PiInterpreterException(format(
180                     "Missing metadata '%s' in packet-in received from '%s': %s",
181                     inportMetadataName, deviceId, packetIn));
182         }
183 
184         // Build ONOS InboundPacket instance with the given ingress port.
185 
186         // 1. Parse packet-in object into Ethernet packet instance.
187         final byte[] payloadBytes = packetIn.data().asArray();
188         final ByteBuffer rawData = ByteBuffer.wrap(payloadBytes);
189         final Ethernet ethPkt;
190         try {
191             ethPkt = Ethernet.deserializer().deserialize(
192                     payloadBytes, 0, packetIn.data().size());
193         } catch (DeserializationException dex) {
194             throw new PiInterpreterException(dex.getMessage());
195         }
196 
197         // 2. Get ingress port
198         final ImmutableByteSequence portBytes = inportMetadata.get().value();
199         final short portNum = portBytes.asReadOnlyBuffer().getShort();
200         final ConnectPoint receivedFrom = new ConnectPoint(
201                 deviceId, PortNumber.portNumber(portNum));
202 
203         return new DefaultInboundPacket(receivedFrom, ethPkt, rawData);
204     }
205 
206     @Override
207     public Optional<Integer> mapLogicalPortNumber(PortNumber port) {
208         if (CONTROLLER.equals(port)) {
209             return Optional.of(CPU_PORT_ID);
210         } else {
211             return Optional.empty();
212         }
213     }
214 
215     @Override
216     public Optional<PiMatchFieldId> mapCriterionType(Criterion.Type type) {
217         if (CRITERION_MAP.containsKey(type)) {
218             return Optional.of(PiMatchFieldId.of(CRITERION_MAP.get(type)));
219         } else {
220             return Optional.empty();
221         }
222     }
223 
224     @Override
225     public PiAction mapTreatment(TrafficTreatment treatment, PiTableId piTableId)
226             throws PiInterpreterException {
227         throw new PiInterpreterException("Treatment mapping not supported");
228     }
229 
230     @Override
231     public Optional<PiTableId> mapFlowRuleTableId(int flowRuleTableId) {
232         return Optional.empty();
233     }
234 }
```

PipelinerImpl.java

```
1  package org.onosproject.ngsdn.tutorial.pipeconf;
2  
3  import org.onosproject.net.DeviceId;
4  import org.onosproject.net.PortNumber;
5  import org.onosproject.net.behaviour.NextGroup;
6  import org.onosproject.net.behaviour.Pipeliner;
7  import org.onosproject.net.behaviour.PipelinerContext;
8  import org.onosproject.net.driver.AbstractHandlerBehaviour;
9  import org.onosproject.net.flow.DefaultFlowRule;
10 import org.onosproject.net.flow.DefaultTrafficTreatment;
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.FlowRuleService;
import org.onosproject.net.flow.instructions.Instructions;
import org.onosproject.net.flowobjective.FilteringObjective;
import org.onosproject.net.flowobjective.ForwardingObjective;
import org.onosproject.net.flowobjective.NextObjective;
import org.onosproject.net.flowobjective.ObjectiveError;
import org.onosproject.net.group.GroupDescription;
import org.onosproject.net.group.GroupService;
import org.onosproject.net.pi.model.PiActionId;
import org.onosproject.net.pi.model.PiTableId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.ngsdn.tutorial.common.Utils;
import org.slf4j.Logger;

import java.util.Collections;
import java.util.List;

import static org.onosproject.net.flow.instructions.Instruction.Type.OUTPUT;
import static org.onosproject.ngsdn.tutorial.AppConstants.CPU_CLONE_SESSION_ID;
import static org.slf4j.LoggerFactory.getLogger;

/**
 * Pipeliner implementation that maps all forwarding objectives to the ACL
 * table. All other types of objectives are not supported.
 */
public class PipelinerImpl extends AbstractHandlerBehaviour implements Pipeliner {

    // From the P4Info file
    private static final String ACL_TABLE = "IngressPipeImpl.acl_table";
    private static final String CLONE_TO_CPU = "IngressPipeImpl.clone_to_cpu";

    private final Logger log = getLogger(getClass());

    private FlowRuleService flowRuleService;
    private GroupService groupService;
    private DeviceId deviceId;


    @Override
    public void init(DeviceId deviceId, PipelinerContext context) {
        this.deviceId = deviceId;
        this.flowRuleService = context.directory().get(FlowRuleService.class);
        this.groupService = context.directory().get(GroupService.class);
    }

    @Override
    public void filter(FilteringObjective obj) {
        obj.context().ifPresent(c -> c.onError(obj, ObjectiveError.UNSUPPORTED));
    }

    @Override
    public void forward(ForwardingObjective obj) {
        if (obj.treatment() == null) {
            obj.context().ifPresent(c -> c.onError(obj, ObjectiveError.UNSUPPORTED));
        }

        // Whether this objective specifies an OUTPUT:CONTROLLER instruction.
        final boolean hasCloneToCpuAction = obj.treatment()
                .allInstructions().stream()
                .filter(i -> i.type().equals(OUTPUT))
                .map(i -> (Instructions.OutputInstruction) i)
                .anyMatch(i -> i.port().equals(PortNumber.CONTROLLER));

        if (!hasCloneToCpuAction) {
            // We support only objectives for clone to CPU behaviours (e.g. for
            // host and link discovery)
            obj.context().ifPresent(c -> c.onError(obj, ObjectiveError.UNSUPPORTED));
        }

        // Create an equivalent FlowRule with same selector and clone_to_cpu action.
        final PiAction cloneToCpuAction = PiAction.builder()
                .withId(PiActionId.of(CLONE_TO_CPU))
                .build();
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
        final FlowRule.Builder ruleBuilder = DefaultFlowRule.builder()
                .forTable(PiTableId.of(ACL_TABLE))
                .forDevice(deviceId)
                .withSelector(obj.selector())
                .fromApp(obj.appId())
                .withPriority(obj.priority())
                .withTreatment(DefaultTrafficTreatment.builder()
                                       .piTableAction(cloneToCpuAction).build());

        if (obj.permanent()) {
            ruleBuilder.makePermanent();
        } else {
            ruleBuilder.makeTemporary(obj.timeout());
        }

        final GroupDescription cloneGroup = Utils.buildCloneGroup(
                obj.appId(),
                deviceId,
                CPU_CLONE_SESSION_ID,
                // Ports where to clone the packet.
                // Just controller in this case.
                Collections.singleton(PortNumber.CONTROLLER));

        switch (obj.op()) {
            case ADD:
                flowRuleService.applyFlowRules(ruleBuilder.build());
                groupService.addGroup(cloneGroup);
                break;
            case REMOVE:
                flowRuleService.removeFlowRules(ruleBuilder.build());
                // Do not remove the clone group as other flow rules might be
                // pointing to it.
                break;
            default:
                log.warn("Unknown operation {}", obj.op());
        }

        obj.context().ifPresent(c -> c.onSuccess(obj));
    }

    @Override
    public void next(NextObjective obj) {
        obj.context().ifPresent(c -> c.onError(obj, ObjectiveError.UNSUPPORTED));
    }

    @Override
    public List<String> getNextMappings(NextGroup nextGroup) {
        // We do not use nextObjectives or groups.
        return Collections.emptyList();
    }
}
```

The `PipelineInterpreter` is the ONOS driver behavior used to map the ONOS representation of packet-in/out to one that is consistent with the P4 pipeline.

Specifically, to use services like link and host discovery, ONOS built-in apps need to be able to set the output port of a packet-out and access the original ingress port of a packet-in.

To trigger device and link discovery the network configuration need to be pushed to ONOS. The network configuration is in JSON format like so:

netcfg.json

```
{
  "devices": {
    "device:leaf1": {
      "basic": {
        "managementAddress": "grpc://mininet:50001?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.ngsdn-tutorial",
        "locType": "grid",
        "gridX": 200,
        "gridY": 600
      },
      "fabricDeviceConfig": {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
        "myStationMac": "00:aa:00:00:00:01",
        "mySid": "3:101:2::",
        "isSpine": false
      }
    },
    "device:leaf2": {
      "basic": {
        "managementAddress": "grpc://mininet:50002?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.ngsdn-tutorial",
        "locType": "grid",
        "gridX": 800,
        "gridY": 600
      },
      "fabricDeviceConfig": {
        "myStationMac": "00:aa:00:00:00:02",
        "mySid": "3:102:2::",
        "isSpine": false
      }
    },
    "device:spine1": {
      "basic": {
        "managementAddress": "grpc://mininet:50003?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.ngsdn-tutorial",
        "locType": "grid",
        "gridX": 400,
        "gridY": 400
      },
      "fabricDeviceConfig": {
        "myStationMac": "00:bb:00:00:00:01",
        "mySid": "3:201:2::",
        "isSpine": true
      }
    },
    "device:spine2": {
      "basic": {
        "managementAddress": "grpc://mininet:50004?device_id=1",
        "driver": "stratum-bmv2",
        "pipeconf": "org.onosproject.ngsdn-tutorial",
        "locType": "grid",
        "gridX": 600,
        "gridY": 400
      },
      "fabricDeviceConfig": {
        "myStationMac": "00:bb:00:00:00:02",
        "mySid": "3:202:2::",
        "isSpine": true
      }
    }
  },
  "ports": {
    "device:leaf1/3": {
      "interfaces": [
        {
          "name": "leaf1-3",
          "ips": ["2001:1:1::ff/64"]
        }
      ]
    },
    "device:leaf1/4": {
      "interfaces": [
        {
          "name": "leaf1-4",
          "ips": ["2001:1:1::ff/64"]
        }
      ]
    },
    "device:leaf1/5": {
      "interfaces": [
        {
          "name": "leaf1-5",
          "ips": ["2001:1:1::ff/64"]
        }
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
      ]
    },
    "device:leaf1/6": {
      "interfaces": [
        {
          "name": "leaf1-6",
          "ips": ["2001:1:2::ff/64"]
        }
      ]
    },
    "device:leaf2/3": {
      "interfaces": [
        {
          "name": "leaf2-3",
          "ips": ["2001:2:3::ff/64"]
        }
      ]
    },
    "device:leaf2/4": {
      "interfaces": [
        {
          "name": "leaf2-4",
          "ips": ["2001:2:4::ff/64"]
        }
      ]
    }
  },
  "hosts": {
    "00:00:00:00:00:1A/None": {
      "basic": {
        "name": "h1a",
        "locType": "grid",
        "gridX": 100,
        "gridY": 700
      }
    },
    "00:00:00:00:00:1B/None": {
      "basic": {
        "name": "h1b",
        "locType": "grid",
        "gridX": 100,
        "gridY": 800
      }
    },
    "00:00:00:00:00:1C/None": {
      "basic": {
        "name": "h1c",
        "locType": "grid",
        "gridX": 250,
        "gridY": 800
      }
    },
    "00:00:00:00:00:20/None": {
      "basic": {
        "name": "h2",
        "locType": "grid",
        "gridX": 400,
        "gridY": 700
      }
    },
    "00:00:00:00:00:30/None": {
      "basic": {
        "name": "h3",
        "locType": "grid",
        "gridX": 750,
        "gridY": 700
      }
    },
    "00:00:00:00:00:40/None": {
      "basic": {
        "name": "h4",
        "locType": "grid",
        "gridX": 850,
        "gridY": 700
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
161        }
162      }
163    }
164  }
```

For verification that all devices and links have been discovered, the ONOS CLI is used:

```
1 onos> devices -s
2 id=device:leaf1, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.onosproject.
    ngsdn-tutorial
3 id=device:leaf2, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.onosproject.
    ngsdn-tutorial
4 id=device:spine1, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.onosproject.
    ngsdn-tutorial
5 id=device:spine2, available=true, role=MASTER, type=SWITCH, driver=stratum-bmv2:org.onosproject.
    ngsdn-tutorial
```

```
1 onos> links
2 src=device:leaf1/1, dst=device:spine1/1, type=DIRECT, state=ACTIVE, expected=false
3 src=device:leaf1/2, dst=device:spine2/1, type=DIRECT, state=ACTIVE, expected=false
4 src=device:leaf2/1, dst=device:spine1/2, type=DIRECT, state=ACTIVE, expected=false
5 src=device:leaf2/2, dst=device:spine2/2, type=DIRECT, state=ACTIVE, expected=false
6 src=device:spine1/1, dst=device:leaf1/1, type=DIRECT, state=ACTIVE, expected=false
7 src=device:spine1/2, dst=device:leaf2/1, type=DIRECT, state=ACTIVE, expected=false
8 src=device:spine2/1, dst=device:leaf1/2, type=DIRECT, state=ACTIVE, expected=false
9 src=device:spine2/2, dst=device:leaf2/2, type=DIRECT, state=ACTIVE, expected=false
```

There should be five flow rules for each device. To show all flow rules installed on device leaf1:

```
1 onos> flows -s any device:leaf1
2 deviceId=device:leaf1, flowRuleCount=5
3     ADDED, ..., table=IngressPipeImpl.acl_table, priority=40000, selector=[ETH_TYPE:lldp],
      treatment=[immediate=[IngressPipeImpl.clone_to_cpu()]]
4     ADDED, ..., table=IngressPipeImpl.acl_table, priority=40000, selector=[ETH_TYPE:bddp],
      treatment=[immediate=[IngressPipeImpl.clone_to_cpu()]]
5     ADDED, ..., table=IngressPipeImpl.acl_table, priority=40000, selector=[ETH_TYPE:arp], treatment
      =[immediate=[IngressPipeImpl.clone_to_cpu()]]
6     ADDED, ..., table=IngressPipeImpl.acl_table, priority=40000, selector=[ETH_TYPE:ipv6, IP_PROTO
      :58, ICMPV6_TYPE:136], treatment=[immediate=[IngressPipeImpl.clone_to_cpu()]]
7     ADDED, ..., table=IngressPipeImpl.acl_table, priority=40000, selector=[ETH_TYPE:ipv6, IP_PROTO
      :58, ICMPV6_TYPE:135], treatment=[immediate=[IngressPipeImpl.clone_to_cpu()]]
8
9     ...
```

These flow rules are the result of the translation of flow objectives generated by the *hostprovider* and *lldp-provider* built-in apps.

Flow objectives are translated by the pipeconf, which provides a Pipeliner behavior implementation (*PipelinerImpl.java*). These flow rules specify a match key by using ONOS standard/known header fields, such as ETH_TYPE, ICMPV6_TYPE, etc. These types are mapped to P4Info-specific match fields by the pipeline interpreter (*InterpreterImpl.java* lines 215-222)

The *hostprovider* app provides host discovery capabilities by intercepting ARP (selector=[ETH_TYPE:arp]) and NDP packets (selector=[ETH_TYPE:ipv6, IP_PROTO:58, ICMPV6_TYPE:...]), which are cloned to the controller (treatment=[immediate=[IngressPipeImpl.clone_to_cpu()]]). Similarly, *lldpprovider* generates flow objectives to intercept LLDP (Link Layer Discovery Protocol) and BDDP (Broadcast Domain Discovery Protocol) packets (selector=[ETH_TYPE:lldp] and selector=[ETH_TYPE:bddp]) periodically emitted on all devices' ports as P4Runtime packet-outs, allowing automatic link discovery.

All flow rules refer to P4 action `clone_to_cpu()`, which invokes a v1model-specific primitive to set the clone session ID (*main.p4* lines 338-344)

To actually generate P4Runtime packet-in messages for matched packets, the pipeconf's pipeliner generates a `CLONE` group, internally translated into a `P4RuntimeCloneSessionEntry`, that maps `CPU_CLONE_SESSION_ID` to a set of ports, just the CPU one in this case.

To show all groups installed in ONOS on leaf1:

```
1 onos> groups any device:leaf1
2 deviceId=device:leaf1, groupCount=1
3    id=0x63, state=ADDED, type=CLONE, ..., appId=org.onosproject.core, referenceCount=0
4        id=0x63, bucket=1, ..., weight=-1, actions=[OUTPUT:CONTROLLER]
```

Figure 26 show how links are visualized on the ONOS UI.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

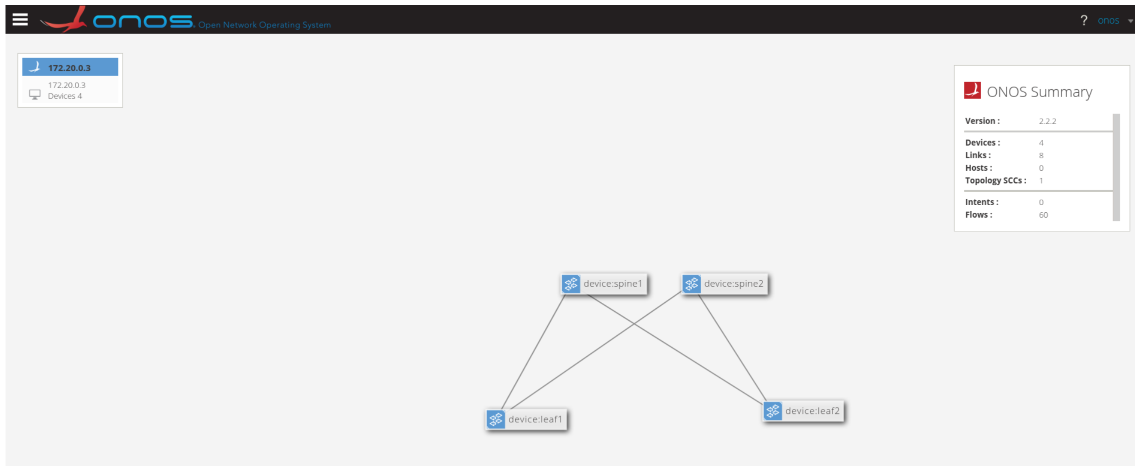


Figure 26: ONOS UI

Link stats are derived by ONOS by periodically obtaining the port counters for each device. ONOS internally uses gNMI to read port information, including counters.

By implementing packet I/O support in the pipeline interpreter we did not only get link discovery, but also enabled the built-in *hostprovider* app to perform host discovery. This service is required by the new app to populate the bridging tables of the P4 pipeline, to forward packets based on the Ethernet destination address.

Indeed, the *hostprovider* app works by snooping incoming ARP/NDP packets on the switch and deducing where a host is connected to from the packet-in message metadata. Other apps in ONOS, like the new app developed, can then listen for host-related events and access information about their addresses (IP, MAC) and location.

6.4.2 L2 Bridging

To implement the L2 bridging a new component app need to be developed. The *L2BridgingComponent.java* app implements and enables L2 bridging.

L2BridgingComponent.java

```
package org.onosproject.ngsdn.tutorial;

import org.onlab.packet.MacAddress;
import org.onosproject.core.ApplicationId;
import org.onosproject.mastership.MastershipService;
import org.onosproject.net.ConnectPoint;
import org.onosproject.net.DeviceId;
import org.onosproject.net.Host;
import org.onosproject.net.PortNumber;
import org.onosproject.net.config.NetworkConfigService;
import org.onosproject.net.device.DeviceEvent;
import org.onosproject.net.device.DeviceListener;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.FlowRuleService;
import org.onosproject.net.flow.criteria.PiCriterion;
import org.onosproject.net.group.GroupDescription;
import org.onosproject.net.group.GroupService;
import org.onosproject.net.host.HostEvent;
import org.onosproject.net.host.HostListener;
import org.onosproject.net.host.HostService;
import org.onosproject.net.intf.Interface;
import org.onosproject.net.intf.InterfaceService;
import org.onosproject.net.pi.model.PiActionId;
import org.onosproject.net.pi.model.PiActionParamId;
import org.onosproject.net.pi.model.PiMatchFieldId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.net.pi.runtime.PiActionParam;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
import org.onosproject.ngsdn.tutorial.common.FabricDeviceConfig;
import org.onosproject.ngsdn.tutorial.common.Utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Set;
import java.util.stream.Collectors;

import static org.onosproject.ngsdn.tutorial.AppConstants.INITIAL_SETUP_DELAY;

/**
 * App component that configures devices to provide L2 bridging capabilities.
 */
@Component(
        immediate = true,
        enabled = true
)
public class L2BridgingComponent {

    private final Logger log = LoggerFactory.getLogger(getClass());

    private static final int DEFAULT_BROADCAST_GROUP_ID = 255;

    private final DeviceListener deviceListener = new InternalDeviceListener();
    private final HostListener hostListener = new InternalHostListener();

    private ApplicationId appId;

    //--------------------------------------------------------------------------
    // ONOS CORE SERVICE BINDING
    //
    // These variables are set by the Karaf runtime environment before calling
    // the activate() method.
    //--------------------------------------------------------------------------

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private HostService hostService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private DeviceService deviceService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private InterfaceService interfaceService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private NetworkConfigService configService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private FlowRuleService flowRuleService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private GroupService groupService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private MastershipService mastershipService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private MainComponent mainComponent;

    //--------------------------------------------------------------------------
    // COMPONENT ACTIVATION.
    //
    // When loading/unloading the app the Karaf runtime environment will call
    // activate()/deactivate().
    //--------------------------------------------------------------------------

    @Activate
    protected void activate() {
        appId = mainComponent.getAppId();

        // Register listeners to be informed about device and host events.
        deviceService.addListener(deviceListener);
        hostService.addListener(hostListener);
        // Schedule set up of existing devices. Needed when reloading the app.
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
        mainComponent.scheduleTask(this::setUpAllDevices, INITIAL_SETUP_DELAY);

        log.info("Started");
    }

    @Deactivate
    protected void deactivate() {
        deviceService.removeListener(deviceListener);
        hostService.removeListener(hostListener);

        log.info("Stopped");
    }

    /**
     * Sets up everything necessary to support L2 bridging on the given device.
     *
     * @param deviceId the device to set up
     */
    private void setUpDevice(DeviceId deviceId) {
        if (isSpine(deviceId)) {
            // Stop here. We support bridging only on leaf/tor switches.
            return;
        }
        insertMulticastGroup(deviceId);
        insertMulticastFlowRules(deviceId);
        // Uncomment the following line after you have implemented the method:
        insertUnmatchedBridgingFlowRule(deviceId);
    }

    /**
     * Inserts an ALL group in the ONOS core to replicate packets on all host
     * facing ports. This group will be used to broadcast all ARP/NDP requests.
     * <p>
     * ALL groups in ONOS are equivalent to P4Runtime packet replication engine
     * (PRE) Multicast groups.
     *
     * @param deviceId the device where to install the group
     */
    private void insertMulticastGroup(DeviceId deviceId) {

        // Replicate packets where we know hosts are attached.
        Set<PortNumber> ports = getHostFacingPorts(deviceId);

        if (ports.isEmpty()) {
            // Stop here.
            log.warn("Device {} has 0 host facing ports", deviceId);
            return;
        }

        log.info("Adding L2 multicast group with {} ports on {}...",
                ports.size(), deviceId);

        // Forge group object.
        final GroupDescription multicastGroup = Utils.buildMulticastGroup(
                appId, deviceId, DEFAULT_BROADCAST_GROUP_ID, ports);

        // Insert.
        groupService.addGroup(multicastGroup);
    }

    /**
     * Insert flow rules matching ethernet destination
     * broadcast/multicast addresses (e.g. ARP requests, NDP Neighbor
     * Solicitation, etc.). Such packets should be processed by the multicast
     * group created before.
     * <p>
     * This method will be called at component activation for each device
     * (switch) known by ONOS, and every time a new device-added event is
     * captured by the InternalDeviceListener defined below.
     *
     * @param deviceId device ID where to install the rules
     */
    private void insertMulticastFlowRules(DeviceId deviceId) {

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
        log.info("Adding L2 multicast rules on {}...", deviceId);

        // Modify P4Runtime entity names to match content of P4Info file (look
        // for the fully qualified name of tables, match fields, and actions.

        // Match ARP request - Match exactly FF:FF:FF:FF:FF:FF
        final PiCriterion macBroadcastCriterion = PiCriterion.builder()
                .matchTernary(
                        PiMatchFieldId.of("hdr.ethernet.dst_addr"),
                        MacAddress.valueOf("FF:FF:FF:FF:FF:FF").toBytes(),
                        MacAddress.valueOf("FF:FF:FF:FF:FF:FF").toBytes())
                .build();

        // Match NDP NS - Match ternary 33:33:**:**:**:**
        final PiCriterion ipv6MulticastCriterion = PiCriterion.builder()
                .matchTernary(
                        PiMatchFieldId.of("hdr.ethernet.dst_addr"),
                        MacAddress.valueOf("33:33:00:00:00:00").toBytes(),
                        MacAddress.valueOf("FF:FF:00:00:00:00").toBytes())
                .build();

        // Action: set multicast group id
        final PiAction setMcastGroupAction = PiAction.builder()
                .withId(PiActionId.of("IngressPipeImpl.set_multicast_group"))
                .withParameter(new PiActionParam(
                        PiActionParamId.of("gid"),
                        DEFAULT_BROADCAST_GROUP_ID))
                .build();

        //  Build 2 flow rules.
        final String tableId = "IngressPipeImpl.l2_ternary_table";

        final FlowRule rule1 = Utils.buildFlowRule(
                deviceId, appId, tableId,
                macBroadcastCriterion, setMcastGroupAction);

        final FlowRule rule2 = Utils.buildFlowRule(
                deviceId, appId, tableId,
                ipv6MulticastCriterion, setMcastGroupAction);

        // Insert rules.
        flowRuleService.applyFlowRules(rule1, rule2);
    }

    /**
     * Insert flow rule that matches all unmatched ethernet traffic. This
     * will implement the traditional briding behavior that floods all
     * unmatched traffic.
     * <p>
     * This method will be called at component activation for each device
     * (switch) known by ONOS, and every time a new device-added event is
     * captured by the InternalDeviceListener defined below.
     *
     * @param deviceId device ID where to install the rules
     */
    @SuppressWarnings("unused")
    private void insertUnmatchedBridgingFlowRule(DeviceId deviceId) {

        log.info("Adding L2 multicast rules on {}...", deviceId);

        // Modify P4Runtime entity names to match content of P4Info file (look
        // for the fully qualified name of tables, match fields, and actions.

        // Match unmatched traffic - Match ternary **:**:**:**:**:**
        final PiCriterion unmatchedTrafficCriterion = PiCriterion.builder()
                .matchTernary(
                        PiMatchFieldId.of("hdr.ethernet.dst_addr"),
                        MacAddress.valueOf("00:00:00:00:00:00").toBytes(),
                        MacAddress.valueOf("00:00:00:00:00:00").toBytes())
                .build();

        // Action: set multicast group id
        final PiAction setMcastGroupAction = PiAction.builder()
                .withId(PiActionId.of("IngressPipeImpl.set_multicast_group"))
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
                .withParameter(new PiActionParam(
                        PiActionParamId.of("gid"),
                        DEFAULT_BROADCAST_GROUP_ID))
                .build();

        //  Build flow rule.
        final String tableId = "IngressPipeImpl.l2_ternary_table";

        final FlowRule rule = Utils.buildFlowRule(
                deviceId, appId, tableId,
                unmatchedTrafficCriterion, setMcastGroupAction);

        // Insert rules.
        flowRuleService.applyFlowRules(rule);
    }

    /**
     * Insert flow rules to forward packets to a given host located at the given
     * device and port.
     * <p>
     * This method will be called at component activation for each host known by
     * ONOS, and every time a new host-added event is captured by the
     * InternalHostListener defined below.
     *
     * @param host     host instance
     * @param deviceId device where the host is located
     * @param port     port where the host is attached to
     */
    private void learnHost(Host host, DeviceId deviceId, PortNumber port) {

        log.info("Adding L2 unicast rule on {} for host {} (port {})...",
                deviceId, host.id(), port);

        // Modify P4Runtime entity names to match content of P4Info file (look
        // for the fully qualified name of tables, match fields, and actions.

        final String tableId = "IngressPipeImpl.l2_exact_table";
        // Match exactly on the host MAC address.
        final MacAddress hostMac = host.mac();
        final PiCriterion hostMacCriterion = PiCriterion.builder()
                .matchExact(PiMatchFieldId.of("hdr.ethernet.dst_addr"),
                        hostMac.toBytes())
                .build();

        // Action: set output port
        final PiAction l2UnicastAction = PiAction.builder()
                .withId(PiActionId.of("IngressPipeImpl.set_egress_port"))
                .withParameter(new PiActionParam(
                        PiActionParamId.of("port_num"),
                        port.toLong()))
                .build();

        // Forge flow rule.
        final FlowRule rule = Utils.buildFlowRule(
                deviceId, appId, tableId, hostMacCriterion, l2UnicastAction);

        // Insert.
        flowRuleService.applyFlowRules(rule);
    }

    //--------------------------------------------------------------------------
    // EVENT LISTENERS
    //
    // Events are processed only if isRelevant() returns true.
    //--------------------------------------------------------------------------

    /**
     * Listener of device events.
     */
    public class InternalDeviceListener implements DeviceListener {

        @Override
        public boolean isRelevant(DeviceEvent event) {
            switch (event.type()) {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
                case DEVICE_ADDED:
                case DEVICE_AVAILABILITY_CHANGED:
                    break;
                default:
                    // Ignore other events.
                    return false;
            }
            // Process only if this controller instance is the master.
            final DeviceId deviceId = event.subject().id();
            return mastershipService.isLocalMaster(deviceId);
        }

        @Override
        public void event(DeviceEvent event) {
            final DeviceId deviceId = event.subject().id();
            if (deviceService.isAvailable(deviceId)) {
                // A P4Runtime device is considered available in ONOS when there
                // is a StreamChannel session open and the pipeline
                // configuration has been set.

                // Events are processed using a thread pool defined in the
                // MainComponent.
                mainComponent.getExecutorService().execute(() -> {
                    log.info("{} event! deviceId={}", event.type(), deviceId);

                    setUpDevice(deviceId);
                });
            }
        }
    }

    /**
     * Listener of host events.
     */
    public class InternalHostListener implements HostListener {

        @Override
        public boolean isRelevant(HostEvent event) {
            switch (event.type()) {
                case HOST_ADDED:
                    // Host added events will be generated by the
                    // HostLocationProvider by intercepting ARP/NDP packets.
                    break;
                case HOST_REMOVED:
                case HOST_UPDATED:
                case HOST_MOVED:
                default:
                    // Ignore other events.
                    // Food for thoughts: how to support host moved/removed?
                    return false;
            }
            // Process host event only if this controller instance is the master
            // for the device where this host is attached to.
            final Host host = event.subject();
            final DeviceId deviceId = host.location().deviceId();
            return mastershipService.isLocalMaster(deviceId);
        }

        @Override
        public void event(HostEvent event) {
            final Host host = event.subject();
            // Device and port where the host is located.
            final DeviceId deviceId = host.location().deviceId();
            final PortNumber port = host.location().port();

            mainComponent.getExecutorService().execute(() -> {
                log.info("{} event! host={}, deviceId={}, port={}",
                        event.type(), host.id(), deviceId, port);

                learnHost(host, deviceId, port);
            });
        }
    }

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
404     //------------------------------------------------------------------------
405     // UTILITY METHODS
406     //------------------------------------------------------------------------
407 
408     /**
409      * Returns a set of ports for the given device that are used to connect
410      * hosts to the fabric.
411      *
412      * @param deviceId device ID
413      * @return set of host facing ports
414      */
415     private Set<PortNumber> getHostFacingPorts(DeviceId deviceId) {
416         // Get all interfaces configured via netcfg for the given device ID and
417         // return the corresponding device port number. Interface configuration
418         // in the netcfg.json looks like this:
419         // "device:leaf1/3": {
420         //   "interfaces": [
421         //     {
422         //       "name": "leaf1-3",
423         //       "ips": ["2001:1:1::ff/64"]
424         //     }
425         //   ]
426         // }
427         return interfaceService.getInterfaces().stream()
428                 .map(Interface::connectPoint)
429                 .filter(cp -> cp.deviceId().equals(deviceId))
430                 .map(ConnectPoint::port)
431                 .collect(Collectors.toSet());
432     }
433 
434     /**
435      * Returns true if the given device is defined as a spine in the
436      * netcfg.json.
437      *
438      * @param deviceId device ID
439      * @return true if spine, false otherwise
440      */
441     private boolean isSpine(DeviceId deviceId) {
442         // Example netcfg defining a device as spine:
443         // "devices": {
444         //   "device:spine1": {
445         //     ...
446         //     "fabricDeviceConfig": {
447         //       "myStationMac": "...",
448         //       "mySid": "...",
449         //       "isSpine": true
450         //     }
451         //   },
452         //   ...
453         final FabricDeviceConfig cfg = configService.getConfig(
454                 deviceId, FabricDeviceConfig.class);
455         return cfg != null && cfg.isSpine();
456     }
457 
458     /**
459      * Sets up L2 bridging on all devices known by ONOS and for which this ONOS
460      * node instance is currently master.
461      * <p>
462      * This method is called at component activation.
463      */
464     private void setUpAllDevices() {
465         deviceService.getAvailableDevices().forEach(device -> {
466             if (mastershipService.isLocalMaster(device.id())) {
467                 log.info("*** L2 BRIDGING - Starting initial set up for {}...", device.id());
468                 setUpDevice(device.id());
469                 // For all hosts connected to this device...
470                 hostService.getConnectedHosts(device.id()).forEach(
471                         host -> learnHost(host, host.location().deviceId(),
472                                 host.location().port()));
473             }
474         });
475     }
476 }
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

This app component defines two event listeners located at the bottom of the `L2BridgingComponent` class, `InternalDeviceListener` (lines 333-367) for device events, e.g. connection of a new switch, and `InternalHostListener` (lines 372-410) for host events, e.g. new host discovered. These listeners in turn call methods like:

- `setUpDevice()` (lines 134-143): responsible for creating multicast groups for all host-facing ports and inserting flow rules for the `l2_ternary_table` pointing to such groups.
- `learnHost()` (lines 292-322): responsible for inserting unicast L2 entries based on the discovered host location.

To support reloading the app implementation, these methods are also called at component activation for all devices and hosts known by ONOS at the time of activation (methods `activate()` and `setUpAllDevices()` in lines 102-113, 134-143 respectively).

To keep things simple, the broadcast domain will be restricted to a single device, i.e. allowing packet replication only for ports of the same leaf switch. As such, ports going to the spines from the multicast group can be excluded.

The *L2BridgingComponent.java* code assumes that hosts of a given subnet are all connected to the same leaf, and two interfaces of two different leaves cannot be configured with the same IPv6 subnet. In other words, L2 bridging is allowed only for hosts connected to the same leaf.

There should be two new flow rules for the `l2_ternary_table` installed by L2BridgingComponent. To show all flow rules installed so far on device leaf1:

```
onos> flows -s any device:leaf1
deviceId=device:leaf1, flowRuleCount=...
    ...
    ADDED, ..., table=IngressPipeImpl.l2_ternary_table, priority=10, selector=[hdr.ethernet.
    dst_addr=0x333300000000&&&0xffff00000000], treatment=[immediate=[IngressPipeImpl.
    set_multicast_group(gid=0xff)]]
    ADDED, ..., table=IngressPipeImpl.l2_ternary_table, priority=10, selector=[hdr.ethernet.
    dst_addr=0xffffffffffff&&&0xffffffffffff], treatment=[immediate=[IngressPipeImpl.
    set_multicast_group(gid=0xff)]]
    ...
```

To show also the multicast groups, the groups command are used. To show groups on leaf1:

```
onos> groups any device:leaf1
deviceId=device:leaf1, groupCount=2
   id=0x63, state=ADDED, type=CLONE, ..., appId=org.onosproject.core, referenceCount=0
       id=0x63, bucket=1, ..., weight=-1, actions=[OUTPUT:CONTROLLER]
   id=0xff, state=ADDED, type=ALL, ..., appId=org.onosproject.ngsdn-tutorial, referenceCount=0
       id=0xff, bucket=1, ..., weight=-1, actions=[OUTPUT:3]
       id=0xff, bucket=2, ..., weight=-1, actions=[OUTPUT:4]
       id=0xff, bucket=3, ..., weight=-1, actions=[OUTPUT:5]
       id=0xff, bucket=4, ..., weight=-1, actions=[OUTPUT:6]
```

The `ALL` group is a new one, created by the app developed (codeappId=org.onosproject.ngsdn-tutorial). Groups of type `ALL` in ONOS map to P4Runtime `MulticastGroupEntry`, in this case used to broadcast NDP NS packets to all host-facing ports.

To verify that L2 bridging works as intended, a ping is sent between hosts in the same subnet, in this case h1a, h1b. After the ping success the hosts should be appeared in ONOS UI as shown in Figure 27. In contrast to what presented at the beginning of this section, here are **not** set any NDP static entries. Instead, NDP NS and NA packets are handled by the data plane thanks to the ALL group and l2_ternary_table's flow rule described above. Moreover, given the ACL flow rules to clone NDP packets to the controller, hosts can be discovered by ONOS. Host discovery events are used by *L2BridgingComponent.java* to insert entries in the P4 `l2_exact_table`. Messages related to the discovery of hosts h1a and h1b are shown checking the ONOS log:

```
INFO  [L2BridgingComponent] HOST_ADDED event! host=00:00:00:00:00:1A/None, deviceId=device:leaf1,
    port=3
INFO  [L2BridgingComponent] Adding L2 unicast rule on device:leaf1 for host 00:00:00:00:00:1A/None
    (port 3)...
INFO  [L2BridgingComponent] HOST_ADDED event! host=00:00:00:00:00:1B/None, deviceId=device:leaf1,
    port=4
INFO  [L2BridgingComponent] Adding L2 unicast rule on device:leaf1 for host 00:00:00:00:00:1B/None
    (port 4).
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

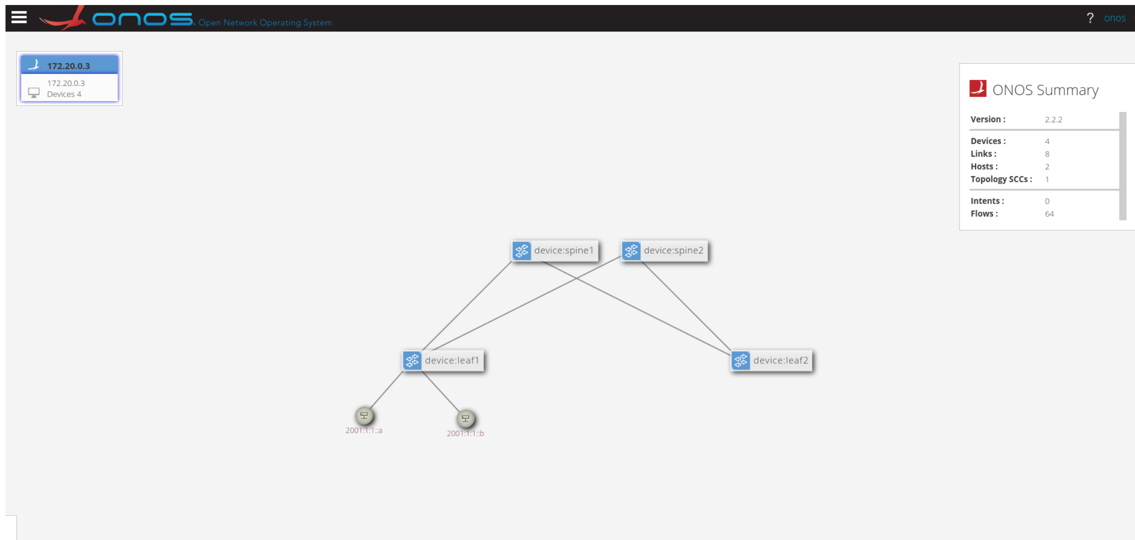


Figure 27: ONOS UI L2 Bridging

6.5 IPv6 Routing

In this subsection, an implementaion for IPv6-based (L3) routing between all hosts connected to the fabric, with support for ECMP to balance traffic flows across multiple spines, is presented.

The first step will be to add new tables to main.p4. The following code is added in the `IngressPipeImpl` block.

```
control IngressPipeImpl (inout parsed_headers_t    hdr,
                         inout local_metadata_t    local_metadata,
                         inout standard_metadata_t standard_metadata) {

    ...

     // --- ndp_reply_table -----------------------------------------------------

    action ndp_ns_to_na(mac_addr_t target_mac) {
        hdr.ethernet.src_addr = target_mac;
        hdr.ethernet.dst_addr = IPV6_MCAST_01;
        ipv6_addr_t host_ipv6_tmp = hdr.ipv6.src_addr;
        hdr.ipv6.src_addr = hdr.ndp.target_ipv6_addr;
        hdr.ipv6.dst_addr = host_ipv6_tmp;
        hdr.ipv6.next_hdr = IP_PROTO_ICMPV6;
        hdr.icmpv6.type = ICMP6_TYPE_NA;
        hdr.ndp.flags = NDP_FLAG_ROUTER | NDP_FLAG_OVERRIDE;
        hdr.ndp.type = NDP_OPT_TARGET_LL_ADDR;
        hdr.ndp.length = 1;
        hdr.ndp.target_mac_addr = target_mac;
        standard_metadata.egress_spec = standard_metadata.ingress_port;
    }

    table ndp_reply_table {
        key = {
            hdr.ndp.target_ipv6_addr: exact;
        }
        actions = {
            ndp_ns_to_na;
        }
        @name("ndp_reply_table_counter")
        counters = direct_counter(CounterType.packets_and_bytes);
    }

    // --- my_station_table ---------------------------------------------------

    table my_station_table {
        key = {
            hdr.ethernet.dst_addr: exact;
        }
        actions = { NoAction; }
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
42          @name("my_station_table_counter")
43          counters = direct_counter(CounterType.packets_and_bytes);
44      }
45
46      // --- routing_v6_table ----------------------------------------------------
47
48      action_selector(HashAlgorithm.crc16, 32w1024, 32w16) ecmp_selector;
49
50      action set_next_hop(mac_addr_t dmac) {
51          hdr.ethernet.src_addr = hdr.ethernet.dst_addr;
52          hdr.ethernet.dst_addr = dmac;
53          // Decrement TTL
54          hdr.ipv6.hop_limit = hdr.ipv6.hop_limit - 1;
55      }
56      table routing_v6_table {
57        key = {
58            hdr.ipv6.dst_addr:          lpm;
59            // The following fields are not used for matching, but as input to the
60            // ecmp_selector hash function.
61            hdr.ipv6.dst_addr:          selector;
62            hdr.ipv6.src_addr:          selector;
63            hdr.ipv6.flow_label:        selector;
64            // The rest of the 5-tuple is optional per RFC6438
65            hdr.ipv6.next_hdr:          selector;
66            local_metadata.l4_src_port: selector;
67            local_metadata.l4_dst_port: selector;
68        }
69        actions = {
70            set_next_hop;
71        }
72        implementation = ecmp_selector;
73        @name("routing_v6_table_counter")
74        counters = direct_counter(CounterType.packets_and_bytes);
75      }
76
77      ...
78
79       apply {
80
81          if (hdr.cpu_out.isValid()) {
82              standard_metadata.egress_spec = hdr.cpu_out.egress_port;
83              hdr.cpu_out.setInvalid();
84              exit;
85          }
86
87          bool do_l3_l2 = true;
88
89          if (hdr.icmpv6.isValid() && hdr.icmpv6.type == ICMP6_TYPE_NS) {
90
91              if (ndp_reply_table.apply().hit) {
92                  do_l3_l2 = false;
93              }
94          }
95
96           if (do_l3_l2) {
97
98              if (hdr.ipv6.isValid() && my_station_table.apply().hit) {
99
100                 routing_v6_table.apply();
101                 // Check TTL, drop packet if necessary to avoid loops.
102                 if(hdr.ipv6.hop_limit == 0) { drop(); }
103             }
104
105             // L2 bridging logic. Apply the exact table first...
106             if (!l2_exact_table.apply().hit) {
107                 // ...if an entry is NOT found, apply the ternary one in case
108                 // this is a multicast/broadcast NDP NS packet.
109                 l2_ternary_table.apply();
110             }
111         }
112
113         // Lastly, apply the ACL table.
114         acl_table.apply();
115     }
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
}
```

We already provide ways to handle NDP NS and NA exchanged by hosts connected to the same subnet. However, for hosts, the Linux networking stack takes care of generating a NDP NA reply. For the switches in this fabric, there is no traditional networking stack associated to it.

There are multiple solutions to this problem:

- configure hosts with static NDP entries, removing the need for the switch to reply to NDP NS packets
- intercept NDP NS via packet-in, generate a corresponding NDP NA reply in ONOS, and send it back via packet-out
- instruct the switch to generate NDP NA replies using P4, i.e., writing P4 code that takes care of replying to NDP requests without any intervention from the control plane

The third approach is selected and implemented.

The idea is simple: NDP NA packets have the same header structure as NDP NS ones. They are both ICMPv6 packets with different header field values, such as different ICMPv6 type, different Ethernet addresses etc. A switch that knows the MAC address of a given IPv6 target address found in an NDP NS request, can transform the same packet to an NDP NA reply by modifying some of its fields.

To implement P4-based generation of NDP NA messages, an action name `ndp_ns_to_na` is implemented to transform an NDP NS packet into an NDP NA one.

This table defines a mapping between the interface IPv6 addresses and the myStationMac associated to each switch, both of which defined in the network configuration JSON file (netcfg.json). When an NDP NS packet is received, asking to resolve one of such IPv6 addresses, the `ndp_ns_to_na` action should be invoked with the given myStationMac as parameter. The ONOS app will be responsible of inserting entries in this table according to the content of the netcfg.json file.

The netcfg.json file includes a special configuration for each device named fabricDeviceConfig, this block defines three values:

- myStationMac: MAC address associated with each device, i.e., the router MAC address
- mySid: the SRv6 segment ID of the device, used later in this section
- isSpine: a boolean flag, indicating whether the device should be considered as a spine switch

Moreover, includes a list of interfaces with an IPv6 prefix assigned to them (lines 69, 77, 85, 93, 101, 109 in netcfg.json).

Two new ONOS components are implemeneted, *Ipv6RoutingComponent.java* and *NdpReplyComponent.java*. The first one provides an implementation for event listeners and the routing policy, methods triggered as a consequence of topology events, for example to compute ECMP groups based on the available links between leaves and the spine. While the latter listens to device events. Each time a new device is added in ONOS, it uses the content of netcfg.json to populate the NDP reply table.

Ipv6RoutingComponent.java

```java
package org.onosproject.ngsdn.tutorial;

import com.google.common.collect.Lists;
import org.onlab.packet.Ip6Address;
import org.onlab.packet.Ip6Prefix;
import org.onlab.packet.IpAddress;
import org.onlab.packet.IpPrefix;
import org.onlab.packet.MacAddress;
import org.onlab.util.ItemNotFoundException;
import org.onosproject.core.ApplicationId;
import org.onosproject.mastership.MastershipService;
import org.onosproject.net.Device;
import org.onosproject.net.DeviceId;
import org.onosproject.net.Host;
import org.onosproject.net.Link;
import org.onosproject.net.PortNumber;
import org.onosproject.net.config.NetworkConfigService;
import org.onosproject.net.device.DeviceEvent;
import org.onosproject.net.device.DeviceListener;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.FlowRuleService;
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
import org.onosproject.net.flow.criteria.PiCriterion;
import org.onosproject.net.group.GroupDescription;
import org.onosproject.net.group.GroupService;
import org.onosproject.net.host.HostEvent;
import org.onosproject.net.host.HostListener;
import org.onosproject.net.host.HostService;
import org.onosproject.net.host.InterfaceIpAddress;
import org.onosproject.net.intf.Interface;
import org.onosproject.net.intf.InterfaceService;
import org.onosproject.net.link.LinkEvent;
import org.onosproject.net.link.LinkListener;
import org.onosproject.net.link.LinkService;
import org.onosproject.net.pi.model.PiActionId;
import org.onosproject.net.pi.model.PiActionParamId;
import org.onosproject.net.pi.model.PiMatchFieldId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.net.pi.runtime.PiActionParam;
import org.onosproject.net.pi.runtime.PiActionProfileGroupId;
import org.onosproject.net.pi.runtime.PiTableAction;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.onosproject.ngsdn.tutorial.common.FabricDeviceConfig;
import org.onosproject.ngsdn.tutorial.common.Utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;

import static com.google.common.collect.Streams.stream;
import static org.onosproject.ngsdn.tutorial.AppConstants.INITIAL_SETUP_DELAY;

/**
 * App component that configures devices to provide IPv6 routing capabilities
 * across the whole fabric.
 */
@Component(
        immediate = true,
        enabled = true
)
public class Ipv6RoutingComponent {

    private static final Logger log = LoggerFactory.getLogger(Ipv6RoutingComponent.class);

    private static final int DEFAULT_ECMP_GROUP_ID = 0xec3b0000;
    private static final long GROUP_INSERT_DELAY_MILLIS = 200;

    private final HostListener hostListener = new InternalHostListener();
    private final LinkListener linkListener = new InternalLinkListener();
    private final DeviceListener deviceListener = new InternalDeviceListener();

    private ApplicationId appId;

    //--------------------------------------------------------------------------
    // ONOS CORE SERVICE BINDING
    //
    // These variables are set by the Karaf runtime environment before calling
    // the activate() method.
    //--------------------------------------------------------------------------

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private FlowRuleService flowRuleService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private HostService hostService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
    private MastershipService mastershipService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private GroupService groupService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private DeviceService deviceService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private NetworkConfigService networkConfigService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private InterfaceService interfaceService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private LinkService linkService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private MainComponent mainComponent;

    //--------------------------------------------------------------------------
    // COMPONENT ACTIVATION.
    //
    // When loading/unloading the app the Karaf runtime environment will call
    // activate()/deactivate().
    //--------------------------------------------------------------------------

    @Activate
    protected void activate() {
        appId = mainComponent.getAppId();

        hostService.addListener(hostListener);
        linkService.addListener(linkListener);
        deviceService.addListener(deviceListener);

        // Schedule set up for all devices.
        mainComponent.scheduleTask(this::setUpAllDevices, INITIAL_SETUP_DELAY);

        log.info("Started");
    }

    @Deactivate
    protected void deactivate() {
        hostService.removeListener(hostListener);
        linkService.removeListener(linkListener);
        deviceService.removeListener(deviceListener);

        log.info("Stopped");
    }

    /**
     * Sets up the "My Station" table for the given device using the
     * myStationMac address found in the config.
     * <p>
     * This method will be called at component activation for each device
     * (switch) known by ONOS, and every time a new device-added event is
     * captured by the InternalDeviceListener defined below.
     *
     * @param deviceId the device ID
     */
    private void setUpMyStationTable(DeviceId deviceId) {

        log.info("Adding My Station rules to {}...", deviceId);

        final MacAddress myStationMac = getMyStationMac(deviceId);

        // the My Station table matches on the *ethernet
        // destination* and there is only one action called *NoAction*, which is
        // used as an indication of "table hit" in the control block.

        final String tableId = "IngressPipeImpl.my_station_table";

        final PiCriterion match = PiCriterion.builder()
                .matchExact(
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
171                         PiMatchFieldId.of("hdr.ethernet.dst_addr"),
172                         myStationMac.toBytes())
173                 .build();
174
175         // Creates an action which do *NoAction* when hit.
176         final PiTableAction action = PiAction.builder()
177                 .withId(PiActionId.of("NoAction"))
178                 .build();
179
180         final FlowRule myStationRule = Utils.buildFlowRule(
181                 deviceId, appId, tableId, match, action);
182
183         flowRuleService.applyFlowRules(myStationRule);
184     }
185
186     /**
187      * Creates an ONOS SELECT group for the routing table to provide ECMP
188      * forwarding for the given collection of next hop MAC addresses. ONOS
189      * SELECT groups are equivalent to P4Runtime action selector groups.
190      * <p>
191      * This method will be called by the routing policy methods below to insert
192      * groups in the L3 table
193      *
194      * @param nextHopMacs the collection of mac addresses of next hops
195      * @param deviceId    the device where the group will be installed
196      * @return a SELECT group
197      */
198     private GroupDescription createNextHopGroup(int groupId,
199                                                 Collection<MacAddress> nextHopMacs,
200                                                 DeviceId deviceId) {
201
202         String actionProfileId = "IngressPipeImpl.ecmp_selector";
203
204         final List<PiAction> actions = Lists.newArrayList();
205
206         // Build one "set next hop" action for each next hop
207
208         final String tableId = "IngressPipeImpl.routing_v6_table";
209         for (MacAddress nextHopMac : nextHopMacs) {
210             final PiAction action = PiAction.builder()
211                     .withId(PiActionId.of("IngressPipeImpl.set_next_hop"))
212                     .withParameter(new PiActionParam(
213                             // Action param name.
214                             PiActionParamId.of("dmac"),
215                             // Action param value.
216                             nextHopMac.toBytes()))
217                     .build();
218
219             actions.add(action);
220         }
221
222         return Utils.buildSelectGroup(
223                 deviceId, tableId, actionProfileId, groupId, actions, appId);
224     }
225
226     /**
227      * Creates a routing flow rule that matches on the given IPv6 prefix and
228      * executes the given group ID (created before).
229      *
230      * @param deviceId  the device where flow rule will be installed
231      * @param ip6Prefix the IPv6 prefix
232      * @param groupId   the group ID
233      * @return a flow rule
234      */
235     private FlowRule createRoutingRule(DeviceId deviceId, Ip6Prefix ip6Prefix,
236                                        int groupId) {
237
238
239         final String tableId = "IngressPipeImpl.routing_v6_table";
240         final PiCriterion match = PiCriterion.builder()
241                 .matchLpm(
242                         PiMatchFieldId.of("hdr.ipv6.dst_addr"),
243                         ip6Prefix.address().toOctets(),
244                         ip6Prefix.prefixLength())
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
                .build();

        final PiTableAction action = PiActionProfileGroupId.of(groupId);

        return Utils.buildFlowRule(
                deviceId, appId, tableId, match, action);
    }

    /**
     * Creates a flow rule for the L2 table mapping the given next hop MAC to
     * the given output port.
     * <p>
     * This is called by the routing policy methods below to establish L2-based
     * forwarding inside the fabric, e.g., when deviceId is a leaf switch and
     * nextHopMac is the one of a spine switch.
     *
     * @param deviceId   the device
     * @param nexthopMac the next hop (destination) mac
     * @param outPort    the output port
     */
    private FlowRule createL2NextHopRule(DeviceId deviceId, MacAddress nexthopMac,
                                         PortNumber outPort) {

        final String tableId = "IngressPipeImpl.l2_exact_table";
        final PiCriterion match = PiCriterion.builder()
                .matchExact(PiMatchFieldId.of("hdr.ethernet.dst_addr"),
                        nexthopMac.toBytes())
                .build();


        final PiAction action = PiAction.builder()
                .withId(PiActionId.of("IngressPipeImpl.set_egress_port"))
                .withParameter(new PiActionParam(
                        PiActionParamId.of("port_num"),
                        outPort.toLong()))
                .build();

        return Utils.buildFlowRule(
                deviceId, appId, tableId, match, action);
    }

    //--------------------------------------------------------------------------
    // EVENT LISTENERS
    //
    // Events are processed only if isRelevant() returns true.
    //--------------------------------------------------------------------------

    /**
     * Listener of host events which triggers configuration of routing rules on
     * the device where the host is attached.
     */
    class InternalHostListener implements HostListener {

        @Override
        public boolean isRelevant(HostEvent event) {
            switch (event.type()) {
                case HOST_ADDED:
                    break;
                case HOST_REMOVED:
                case HOST_UPDATED:
                case HOST_MOVED:
                default:
                    // Ignore other events.
                    // Food for thoughts:
                    // how to support host moved/removed events?
                    return false;
            }
            // Process host event only if this controller instance is the master
            // for the device where this host is attached.
            final Host host = event.subject();
            final DeviceId deviceId = host.location().deviceId();
            return mastershipService.isLocalMaster(deviceId);
        }

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
319         @Override
320         public void event(HostEvent event) {
321             Host host = event.subject();
322             DeviceId deviceId = host.location().deviceId();
323             mainComponent.getExecutorService().execute(() -> {
324                 log.info("{} event! host={}, deviceId={}, port={}",
325                         event.type(), host.id(), deviceId, host.location().port());
326                 setUpHostRules(deviceId, host);
327             });
328         }
329     }
330 
331     /**
332      * Listener of link events, which triggers configuration of routing rules to
333      * forward packets across the fabric, i.e. from leaves to spines and vice
334      * versa.
335      * <p>
336      * Reacting to link events instead of device ones, allows us to make sure
337      * all device are always configured with a topology view that includes all
338      * links, e.g. modifying an ECMP group as soon as a new link is added. The
339      * downside is that we might be configuring the same device twice for the
340      * same set of links/paths. However, the ONOS core treats these cases as a
341      * no-op when the device is already configured with the desired forwarding
342      * state (i.e. flows and groups)
343      */
344     class InternalLinkListener implements LinkListener {
345 
346         @Override
347         public boolean isRelevant(LinkEvent event) {
348             switch (event.type()) {
349                 case LINK_ADDED:
350                     break;
351                 case LINK_UPDATED:
352                 case LINK_REMOVED:
353                 default:
354                     return false;
355             }
356             DeviceId srcDev = event.subject().src().deviceId();
357             DeviceId dstDev = event.subject().dst().deviceId();
358             return mastershipService.isLocalMaster(srcDev) ||
359                     mastershipService.isLocalMaster(dstDev);
360         }
361 
362         @Override
363         public void event(LinkEvent event) {
364             DeviceId srcDev = event.subject().src().deviceId();
365             DeviceId dstDev = event.subject().dst().deviceId();
366 
367             if (mastershipService.isLocalMaster(srcDev)) {
368                 mainComponent.getExecutorService().execute(() -> {
369                     log.info("{} event! Configuring {}... linkSrc={}, linkDst={}",
370                             event.type(), srcDev, srcDev, dstDev);
371                     setUpFabricRoutes(srcDev);
372                     setUpL2NextHopRules(srcDev);
373                 });
374             }
375             if (mastershipService.isLocalMaster(dstDev)) {
376                 mainComponent.getExecutorService().execute(() -> {
377                     log.info("{} event! Configuring {}... linkSrc={}, linkDst={}",
378                             event.type(), dstDev, srcDev, dstDev);
379                     setUpFabricRoutes(dstDev);
380                     setUpL2NextHopRules(dstDev);
381                 });
382             }
383         }
384     }
385 
386     /**
387      * Listener of device events which triggers configuration of the My Station
388      * table.
389      */
390     class InternalDeviceListener implements DeviceListener {
391 
392         @Override
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
        public boolean isRelevant(DeviceEvent event) {
            switch (event.type()) {
                case DEVICE_AVAILABILITY_CHANGED:
                case DEVICE_ADDED:
                    break;
                default:
                    return false;
            }
            // Process device event if this controller instance is the master
            // for the device and the device is available.
            DeviceId deviceId = event.subject().id();
            return mastershipService.isLocalMaster(deviceId) &&
                    deviceService.isAvailable(event.subject().id());
        }

        @Override
        public void event(DeviceEvent event) {
            mainComponent.getExecutorService().execute(() -> {
                DeviceId deviceId = event.subject().id();
                log.info("{} event! device id={}", event.type(), deviceId);
                setUpMyStationTable(deviceId);
            });
        }
    }

    //--------------------------------------------------------------------------
    // ROUTING POLICY METHODS
    //
    // Called by event listeners, these methods implement the actual routing
    // policy, responsible of computing paths and creating ECMP groups.
    //--------------------------------------------------------------------------

    /**
     * Set up L2 nexthop rules of a device to providing forwarding inside the
     * fabric, i.e. between leaf and spine switches.
     *
     * @param deviceId the device ID
     */
    private void setUpL2NextHopRules(DeviceId deviceId) {

        Set<Link> egressLinks = linkService.getDeviceEgressLinks(deviceId);

        for (Link link : egressLinks) {
            // For each other switch directly connected to this.
            final DeviceId nextHopDevice = link.dst().deviceId();
            // Get port of this device connecting to next hop.
            final PortNumber outPort = link.src().port();
            // Get next hop MAC address.
            final MacAddress nextHopMac = getMyStationMac(nextHopDevice);

            final FlowRule nextHopRule = createL2NextHopRule(
                    deviceId, nextHopMac, outPort);

            flowRuleService.applyFlowRules(nextHopRule);
        }
    }

    /**
     * Sets up the given device with the necessary rules to route packets to the
     * given host.
     *
     * @param deviceId deviceId the device ID
     * @param host     the host
     */
    private void setUpHostRules(DeviceId deviceId, Host host) {

        // Get all IPv6 addresses associated to this host. In this tutorial we
        // use hosts with only 1 IPv6 address.
        final Collection<Ip6Address> hostIpv6Addrs = host.ipAddresses().stream()
                .filter(IpAddress::isIp6)
                .map(IpAddress::getIp6Address)
                .collect(Collectors.toSet());

        if (hostIpv6Addrs.isEmpty()) {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
467            // Ignore.
468            log.debug("No IPv6 addresses for host {}, ignore", host.id());
469            return;
470        } else {
471            log.info("Adding routes on {} for host {} [{}]",
472                    deviceId, host.id(), hostIpv6Addrs);
473        }
474
475        // Create an ECMP group with only one member, where the group ID is
476        // derived from the host MAC.
477        final MacAddress hostMac = host.mac();
478        int groupId = macToGroupId(hostMac);
479
480        final GroupDescription group = createNextHopGroup(
481                groupId, Collections.singleton(hostMac), deviceId);
482
483        // Map each host IPV6 address to corresponding /128 prefix and obtain a
484        // flow rule that points to the group ID. In this tutorial we expect
485        // only one flow rule per host.
486        final List<FlowRule> flowRules = hostIpv6Addrs.stream()
487                .map(IpAddress::toIpPrefix)
488                .filter(IpPrefix::isIp6)
489                .map(IpPrefix::getIp6Prefix)
490                .map(prefix -> createRoutingRule(deviceId, prefix, groupId))
491                .collect(Collectors.toList());
492
493        // Helper function to install flows after groups, since here flows
494        // points to the group and P4Runtime enforces this dependency during
495        // write operations.
496        insertInOrder(group, flowRules);
497    }
498
499    /**
500     * Set up routes on a given device to forward packets across the fabric,
501     * making a distinction between spines and leaves.
502     *
503     * @param deviceId the device ID.
504     */
505    private void setUpFabricRoutes(DeviceId deviceId) {
506        if (isSpine(deviceId)) {
507            setUpSpineRoutes(deviceId);
508        } else {
509            setUpLeafRoutes(deviceId);
510        }
511    }
512
513    /**
514     * Insert routing rules on the given spine switch, matching on leaf
515     * interface subnets and forwarding packets to the corresponding leaf.
516     *
517     * @param spineId the spine device ID
518     */
519    private void setUpSpineRoutes(DeviceId spineId) {
520
521        log.info("Adding up spine routes on {}...", spineId);
522
523        for (Device device : deviceService.getDevices()) {
524
525            if (isSpine(device.id())) {
526                // We only need routes to leaf switches. Ignore spines.
527                continue;
528            }
529
530            final DeviceId leafId = device.id();
531            final MacAddress leafMac = getMyStationMac(leafId);
532            final Set<Ip6Prefix> subnetsToRoute = getInterfaceIpv6Prefixes(leafId);
533
534            final Ip6Address leafSid = getDeviceSid(leafId);
535            subnetsToRoute.add(Ip6Prefix.valueOf(leafSid, 128));
536
537            // Create a group with only one member.
538            int groupId = macToGroupId(leafMac);
539
540            GroupDescription group = createNextHopGroup(
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
                groupId, Collections.singleton(leafMac), spineId);

        List<FlowRule> flowRules = subnetsToRoute.stream()
                .map(subnet -> createRoutingRule(spineId, subnet, groupId))
                .collect(Collectors.toList());

        insertInOrder(group, flowRules);
    }
}

/**
 * Insert routing rules on the given leaf switch, matching on interface
 * subnets associated to other leaves and forwarding packets the spines
 * using ECMP.
 *
 * @param leafId the leaf device ID
 */
private void setUpLeafRoutes(DeviceId leafId) {
    log.info("Setting up leaf routes: {}", leafId);

    // Get the set of subnets (interface IPv6 prefixes) associated to other
    // leafs but not this one.
    Set<Ip6Prefix> subnetsToRouteViaSpines = stream(deviceService.getDevices())
            .map(Device::id)
            .filter(this::isLeaf)
            .filter(deviceId -> !deviceId.equals(leafId))
            .map(this::getInterfaceIpv6Prefixes)
            .flatMap(Collection::stream)
            .collect(Collectors.toSet());

    // Get myStationMac address of all spines.
    Set<MacAddress> spineMacs = stream(deviceService.getDevices())
            .map(Device::id)
            .filter(this::isSpine)
            .map(this::getMyStationMac)
            .collect(Collectors.toSet());

    // Create an ECMP group to distribute traffic across all spines.
    final int groupId = DEFAULT_ECMP_GROUP_ID;
    final GroupDescription ecmpGroup = createNextHopGroup(
            groupId, spineMacs, leafId);

    // Generate a flow rule for each subnet pointing to the ECMP group.
    List<FlowRule> flowRules = subnetsToRouteViaSpines.stream()
            .map(subnet -> createRoutingRule(leafId, subnet, groupId))
            .collect(Collectors.toList());

    insertInOrder(ecmpGroup, flowRules);

    stream(deviceService.getDevices())
            .map(Device::id)
            .filter(this::isSpine)
            .forEach(spineId -> {
                MacAddress spineMac = getMyStationMac(spineId);
                Ip6Address spineSid = getDeviceSid(spineId);
                int spineGroupId = macToGroupId(spineMac);
                GroupDescription group = createNextHopGroup(
                        spineGroupId, Collections.singleton(spineMac), leafId);
                FlowRule routingRule = createRoutingRule(
                        leafId, Ip6Prefix.valueOf(spineSid, 128),
                        spineGroupId);
                insertInOrder(group, Collections.singleton(routingRule));
            });
}

//--------------------------------------------------------------------------
// UTILITY METHODS
//--------------------------------------------------------------------------

/**
 * Returns true if the given device has isSpine flag set to true in the
 * config, false otherwise.
 *
 * @param deviceId the device ID
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
615      * @return true if the device is a spine, false otherwise
616      */
617     private boolean isSpine(DeviceId deviceId) {
618         return getDeviceConfig(deviceId).map(FabricDeviceConfig::isSpine)
619                 .orElseThrow(() -> new ItemNotFoundException(
620                         "Missing isSpine config for " + deviceId));
621     }
622 
623     /**
624      * Returns true if the given device is not configured as spine.
625      *
626      * @param deviceId the device ID
627      * @return true if the device is a leaf, false otherwise
628      */
629     private boolean isLeaf(DeviceId deviceId) {
630         return !isSpine(deviceId);
631     }
632 
633     /**
634      * Returns the MAC address configured in the "myStationMac" property of the
635      * given device config.
636      *
637      * @param deviceId the device ID
638      * @return MyStation MAC address
639      */
640     private MacAddress getMyStationMac(DeviceId deviceId) {
641         return getDeviceConfig(deviceId)
642                 .map(FabricDeviceConfig::myStationMac)
643                 .orElseThrow(() -> new ItemNotFoundException(
644                         "Missing myStationMac config for " + deviceId));
645     }
646 
647     /**
648      * Returns the FabricDeviceConfig config object for the given device.
649      *
650      * @param deviceId the device ID
651      * @return FabricDeviceConfig device config
652      */
653     private Optional<FabricDeviceConfig> getDeviceConfig(DeviceId deviceId) {
654         FabricDeviceConfig config = networkConfigService.getConfig(
655                 deviceId, FabricDeviceConfig.class);
656         return Optional.ofNullable(config);
657     }
658 
659     /**
660      * Returns the set of interface IPv6 subnets (prefixes) configured for the
661      * given device.
662      *
663      * @param deviceId the device ID
664      * @return set of IPv6 prefixes
665      */
666     private Set<Ip6Prefix> getInterfaceIpv6Prefixes(DeviceId deviceId) {
667         return interfaceService.getInterfaces().stream()
668                 .filter(iface -> iface.connectPoint().deviceId().equals(deviceId))
669                 .map(Interface::ipAddressesList)
670                 .flatMap(Collection::stream)
671                 .map(InterfaceIpAddress::subnetAddress)
672                 .filter(IpPrefix::isIp6)
673                 .map(IpPrefix::getIp6Prefix)
674                 .collect(Collectors.toSet());
675     }
676 
677     /**
678      * Returns a 32 bit bit group ID from the given MAC address.
679      *
680      * @param mac the MAC address
681      * @return an integer
682      */
683     private int macToGroupId(MacAddress mac) {
684         return mac.hashCode() & 0x7fffffff;
685     }
686 
687     /**
688      * Inserts the given groups and flow rules in order, groups first, then flow
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
     * rules. In P4Runtime, when operating on an indirect table (i.e. with
     * action selectors), groups must be inserted before table entries.
     *
     * @param group     the group
     * @param flowRules the flow rules depending on the group
     */
    private void insertInOrder(GroupDescription group, Collection<FlowRule> flowRules) {
        try {
            groupService.addGroup(group);
            // Wait for groups to be inserted.
            Thread.sleep(GROUP_INSERT_DELAY_MILLIS);
            flowRules.forEach(flowRuleService::applyFlowRules);
        } catch (InterruptedException e) {
            log.error("Interrupted!", e);
            Thread.currentThread().interrupt();
        }
    }

    /**
     * Gets Srv6 SID for the given device.
     *
     * @param deviceId the device ID
     * @return SID for the device
     */
    private Ip6Address getDeviceSid(DeviceId deviceId) {
        return getDeviceConfig(deviceId)
                .map(FabricDeviceConfig::mySid)
                .orElseThrow(() -> new ItemNotFoundException(
                        "Missing mySid config for " + deviceId));
    }

    /**
     * Sets up IPv6 routing on all devices known by ONOS and for which this ONOS
     * node instance is currently master.
     */
    private synchronized void setUpAllDevices() {
        // Set up host routes
        stream(deviceService.getAvailableDevices())
                .map(Device::id)
                .filter(mastershipService::isLocalMaster)
                .forEach(deviceId -> {
                    log.info("*** IPV6 ROUTING - Starting initial set up for {}...", deviceId);
                    setUpMyStationTable(deviceId);
                    setUpFabricRoutes(deviceId);
                    setUpL2NextHopRules(deviceId);
                    hostService.getConnectedHosts(deviceId)
                            .forEach(host -> setUpHostRules(deviceId, host));
                });
    }
}
```

NdpReplyComponent.java

```
package org.onosproject.ngsdn.tutorial;

import org.onlab.packet.Ip6Address;
import org.onlab.packet.IpAddress;
import org.onlab.packet.MacAddress;
import org.onlab.util.ItemNotFoundException;
import org.onosproject.core.ApplicationId;
import org.onosproject.mastership.MastershipService;
import org.onosproject.net.DeviceId;
import org.onosproject.net.config.NetworkConfigService;
import org.onosproject.net.device.DeviceEvent;
import org.onosproject.net.device.DeviceListener;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.FlowRuleOperations;
import org.onosproject.net.flow.FlowRuleService;
import org.onosproject.net.flow.criteria.PiCriterion;
import org.onosproject.net.host.InterfaceIpAddress;
import org.onosproject.net.intf.Interface;
import org.onosproject.net.intf.InterfaceService;
import org.onosproject.net.pi.model.PiActionId;
import org.onosproject.net.pi.model.PiActionParamId;
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
import org.onosproject.net.pi.model.PiMatchFieldId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.net.pi.runtime.PiActionParam;
import org.onosproject.ngsdn.tutorial.common.FabricDeviceConfig;
import org.onosproject.ngsdn.tutorial.common.Utils;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collection;
import java.util.stream.Collectors;

import static org.onosproject.ngsdn.tutorial.AppConstants.INITIAL_SETUP_DELAY;

/**
 * App component that configures devices to generate NDP Neighbor Advertisement
 * packets for all interface IPv6 addresses configured in the netcfg.
 */
@Component(
        immediate = true,
        enabled = true
)
public class NdpReplyComponent {

    private static final Logger log =
            LoggerFactory.getLogger(NdpReplyComponent.class.getName());

    //--------------------------------------------------------------------------
    // ONOS CORE SERVICE BINDING
    //
    // These variables are set by the Karaf runtime environment before calling
    // the activate() method.
    //--------------------------------------------------------------------------

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected NetworkConfigService configService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected FlowRuleService flowRuleService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected InterfaceService interfaceService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected MastershipService mastershipService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected DeviceService deviceService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private MainComponent mainComponent;

    private DeviceListener deviceListener = new InternalDeviceListener();
    private ApplicationId appId;

    //--------------------------------------------------------------------------
    // COMPONENT ACTIVATION.
    //
    // When loading/unloading the app the Karaf runtime environment will call
    // activate()/deactivate().
    //--------------------------------------------------------------------------

    @Activate
    public void activate() {
        appId = mainComponent.getAppId();
        // Register listeners to be informed about device events.
        deviceService.addListener(deviceListener);
        // Schedule set up of existing devices. Needed when reloading the app.
        mainComponent.scheduleTask(this::setUpAllDevices, INITIAL_SETUP_DELAY);
        log.info("Started");
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
    }

    @Deactivate
    public void deactivate() {
        deviceService.removeListener(deviceListener);
        log.info("Stopped");
    }

    /**
     * Set up all devices for which this ONOS instance is currently master.
     */
    private void setUpAllDevices() {
        deviceService.getAvailableDevices().forEach(device -> {
            if (mastershipService.isLocalMaster(device.id())) {
                log.info("*** NDP REPLY - Starting Initial set up for {}...", device.id());
                setUpDevice(device.id());
            }
        });
    }

    /**
     * Performs setup of the given device by creating a flow rule to generate
     * NDP NA packets for IPv6 addresses associated to the device interfaces.
     *
     * @param deviceId device ID
     */
    private void setUpDevice(DeviceId deviceId) {

        // Get this device config from netcfg.json.
        final FabricDeviceConfig config = configService.getConfig(
                deviceId, FabricDeviceConfig.class);
        if (config == null) {
            // Config not available yet
            throw new ItemNotFoundException("Missing fabricDeviceConfig for " + deviceId);
        }

        // Get this device myStation mac.
        final MacAddress deviceMac = config.myStationMac();

        // Get all interfaces currently configured for the device
        final Collection<Interface> interfaces = interfaceService.getInterfaces()
                .stream()
                .filter(iface -> iface.connectPoint().deviceId().equals(deviceId))
                .collect(Collectors.toSet());

        if (interfaces.isEmpty()) {
            log.info("{} does not have any IPv6 interface configured",
                     deviceId);
            return;
        }

        // Generate and install flow rules.
        log.info("Adding rules to {} to generate NDP NA for {} IPv6 interfaces...",
                 deviceId, interfaces.size());
        final Collection<FlowRule> flowRules = interfaces.stream()
                .map(this::getIp6Addresses)
                .flatMap(Collection::stream)
                .map(ipv6addr -> buildNdpReplyFlowRule(deviceId, ipv6addr, deviceMac))
                .collect(Collectors.toSet());

        installRules(flowRules);
    }

    /**
     * Build a flow rule for the NDP reply table on the given device, for the
     * given target IPv6 address and MAC address.
     *
     * @param deviceId          device ID where to install the flow rules
     * @param targetIpv6Address target IPv6 address
     * @param targetMac         target MAC address
     * @return flow rule object
     */
    private FlowRule buildNdpReplyFlowRule(DeviceId deviceId,
                                           Ip6Address targetIpv6Address,
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
                                                 MacAddress targetMac) {

        // Build match.
        final PiCriterion match = PiCriterion.builder()
                .matchExact(PiMatchFieldId.of("hdr.ndp.target_ipv6_addr"), targetIpv6Address.
    toOctets())
                .build();
        // Build action.
        final PiActionParam targetMacParam = new PiActionParam(
                PiActionParamId.of("target_mac"), targetMac.toBytes());
        final PiAction action = PiAction.builder()
                .withId(PiActionId.of("IngressPipeImpl.ndp_ns_to_na"))
                .withParameter(targetMacParam)
                .build();
        // Table ID.
        final String tableId = "IngressPipeImpl.ndp_reply_table";

        // Build flow rule.
        final FlowRule rule = Utils.buildFlowRule(
                deviceId, appId, tableId, match, action);

        return rule;
    }

    //--------------------------------------------------------------------------
    // EVENT LISTENERS
    //
    // Events are processed only if isRelevant() returns true.
    //--------------------------------------------------------------------------

    /**
     * Listener of device events.
     */
    public class InternalDeviceListener implements DeviceListener {

        @Override
        public boolean isRelevant(DeviceEvent event) {
            switch (event.type()) {
                case DEVICE_ADDED:
                case DEVICE_AVAILABILITY_CHANGED:
                    break;
                default:
                    // Ignore other events.
                    return false;
            }
            // Process only if this controller instance is the master.
            final DeviceId deviceId = event.subject().id();
            return mastershipService.isLocalMaster(deviceId);
        }

        @Override
        public void event(DeviceEvent event) {
            final DeviceId deviceId = event.subject().id();
            if (deviceService.isAvailable(deviceId)) {
                // A P4Runtime device is considered available in ONOS when there
                // is a StreamChannel session open and the pipeline
                // configuration has been set.

                // Events are processed using a thread pool defined in the
                // MainComponent.
                mainComponent.getExecutorService().execute(() -> {
                    log.info("{} event! deviceId={}", event.type(), deviceId);
                    setUpDevice(deviceId);
                });
            }
        }
    }

    //--------------------------------------------------------------------------
    // UTILITY METHODS
    //--------------------------------------------------------------------------

    /**
     * Returns all IPv6 addresses associated with the given interface.
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```java
     *
     * @param iface interface instance
     * @return collection of IPv6 addresses
     */
    private Collection<Ip6Address> getIp6Addresses(Interface iface) {
        return iface.ipAddressesList()
                .stream()
                .map(InterfaceIpAddress::ipAddress)
                .filter(IpAddress::isIp6)
                .map(IpAddress::getIp6Address)
                .collect(Collectors.toSet());
    }

    /**
     * Install the given flow rules in batch using the flow rule service.
     *
     * @param flowRules flow rules to install
     */
    private void installRules(Collection<FlowRule> flowRules) {
        FlowRuleOperations.Builder ops = FlowRuleOperations.builder();
        flowRules.forEach(ops::add);
        flowRuleService.apply(ops.build());
    }
}
```

Pinging between h3 and h2 should work now. The ONOS log should show messages such as:

```
...
INFO  [Ipv6RoutingComponent] HOST_ADDED event! host=00:00:00:00:00:20/None, deviceId=device:leaf1,
    port=6
INFO  [Ipv6RoutingComponent] Adding routes on device:leaf1 for host 00:00:00:00:00:20/None
    [[2001:1:2::1]]
...
INFO  [Ipv6RoutingComponent] HOST_ADDED event! host=00:00:00:00:00:30/None, deviceId=device:leaf2,
    port=3
INFO  [Ipv6RoutingComponent] Adding routes on device:leaf2 for host 00:00:00:00:00:30/None
    [[2001:2:3::1]]
...
```

To verify that the P4-based generation of NDP NA replies by the switch is working, can be checked by inspecting the neighbor table of h2 or h3. It should show something similar to this:

```
mininet> h3 ip -6 n
2001:2:3::ff dev h3-eth0 lladdr 00:aa:00:00:00:02 router REACHABLE
```

where 2001:2:3::ff is the IPv6 gateway address defined in netcfg.json and 00:aa:00:00:00:02 is the myStationMac defined for leaf2 in netcfg.json.

To verify that ECMP is working, is started by multiple parallel traffic flows from h2 to h3 using iperf.

```
mininet> h2 iperf -c h3 -u -V -P5 -b1M -t600 -i1
```

This commands starts an iperf client on h2, sending UDP packets (-u) over IPv6 (-V) to h3 (-c). By doing this, 5 distinct flows (-P5) are generated, each one capped at 1Mbit/s (-b1M), running for 10 minutes (-t600) and reporting stats every 1 second (-i1). Since the generated traffic is UDP, there is no need to start an iperf server on h3.

Figure 28 shows the traffic being forwarded to both spines, meaning that ECMP is working.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

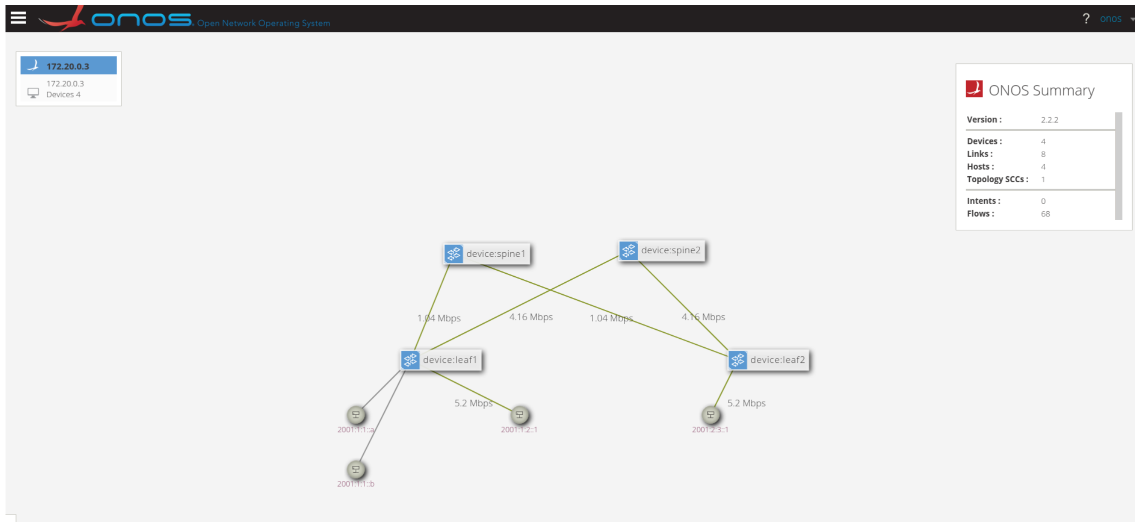


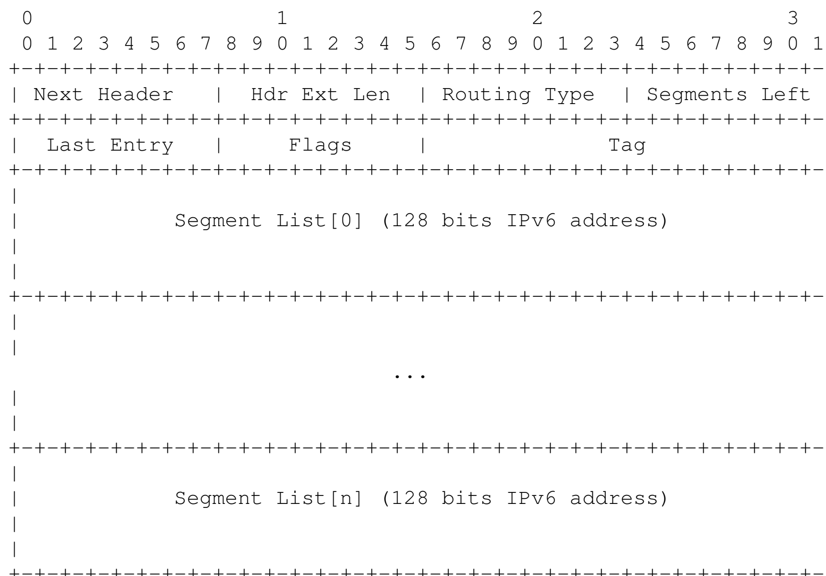
Figure 28: ONOS UI IPv6 Routing with ECMP

6.6 Segment Routing v6

In this subsection, an implementation of a simplified version of segment routing, a source routing method that steers traffic through a specified set of nodes will be presented.

This implementation is based on an IETF draft specification called SRv6, which uses IPv6 packets to frame traffic that follows an SRv6 policy. SRv6 packets use the IPv6 routing header, and they can either encapsulate IPv6 or IPv4 packets entirely or they can just inject an IPv6 routing header into an existing IPv6 packet.

The IPv6 routing header looks as follows:



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Last Entry   |     Flags     |              Tag              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The **Next Header** field is the type of either the next IPv6 header or the payload and for SRv6, the **Routing Type** is 4. **Segments Left** points to the index of the current segment in the segment list. In properly formed SRv6 packets, the IPv6 destination address equals Segment List[Segments Left]. The original IPv6 address should be Segment List[0] in this implementation so that traffic is routed to the correct destination eventually. **Last Entry** is the index of the last entry in the segment list. This means it should be one less than the length of the list. Finally, the **Segment List** is a reverse-sorted list of IPv6 addresses to be traversed in a specific SRv6 policy. The last entry in the list is the first segment in the SRv6 policy. The list is not typically mutated; the entire header is inserted or removed as a whole.

SRv6 uses IPv6 addresses to identify segments in a policy. While the format of the addresses is the same as IPv6, the address space is typically different from the space used for switch's internal IPv6 addresses. The format of the address also differs. A typical IPv6 unicast address is broken into a network prefix and host identifier pieces, and a subnet mask is used to delineate the boundary between the two. A typical SRv6

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

segment identifier (SID) is broken into a locator, a function identifier, and optionally, function arguments. The locator must be routable, which enables both SRv6-enable and unaware nodes to participate in forwarding. Due to optional arguments, longest prefix match on the 128-bit SID is preferred to exact match.

There are three types of nodes of interest in a segment routed network:

- Source Node - the node, either host or switch, that injects the SRv6 policy.
- Transit Node - a node that forwards an SRv6 packet, but is not the destination for the traffic
- Endpoint Node - a participating waypoint in an SRv6 policy that will modify the SRv6 header and perform a specified function

In this implementation, these types are simplified into two roles:

- Endpoint Node - for traffic to the switch's SID, update the SRv6 header (decrement segments left), set the IPv6 destination address to the next segment, and forward the packets ("End" behavior). For simplicity, the SRv6 header is always removed on the penultimate segment in the policy (called Penultimate Segment Pop or PSP in the spec).
- Transit Node - by default, forward traffic normally if it is not destined for the switch's IP address or its SID ("T" behavior). Allow the control plane to add rules to inject SRv6 policy for traffic destined to specific IPv6 addresses ("T.Insert" behavior).

The following code is added to the *main.p4* file.

```
...
// Maximum number of hops supported when using SRv6.
#define SRV6_MAX_HOPS 4

...

const bit<8> IP_PROTO_ICMP   = 1;
const bit<8> IP_PROTO_TCP    = 6;
const bit<8> IP_PROTO_UDP    = 17;
const bit<8> IP_PROTO_SRV6   = 43;
const bit<8> IP_PROTO_ICMPV6 = 58;

...

header srv6h_t {
    bit<8>   next_hdr;
    bit<8>   hdr_ext_len;
    bit<8>   routing_type;
    bit<8>   segment_left;
    bit<8>   last_entry;
    bit<8>   flags;
    bit<16>  tag;
}


header srv6_list_t {
    bit<128>  segment_id;
}

...

struct parsed_headers_t {
    cpu_out_header_t cpu_out;
    cpu_in_header_t cpu_in;
    ethernet_t ethernet;
    ipv4_t ipv4;
    ipv6_t ipv6;
    srv6h_t srv6h;
    srv6_list_t[SRV6_MAX_HOPS] srv6_list;
    tcp_t tcp;
    udp_t udp;
    icmp_t icmp;
    icmpv6_t icmpv6;
    ndp_t ndp;
}

```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
struct local_metadata_t {
    l4_port_t   l4_src_port;
    l4_port_t   l4_dst_port;
    bool        is_multicast;
    ipv6_addr_t next_srv6_sid;
    bit<8>      ip_proto;
    bit<8>      icmp_type;
}

...

parser ParserImpl (packet_in packet,
                   out parsed_headers_t hdr,
                   inout local_metadata_t local_metadata,
                   inout standard_metadata_t standard_metadata)
{

    ...

    state parse_ipv6 {
        packet.extract(hdr.ipv6);
        local_metadata.ip_proto = hdr.ipv6.next_hdr;
        transition select(hdr.ipv6.next_hdr) {
            IP_PROTO_TCP: parse_tcp;
            IP_PROTO_UDP: parse_udp;
            IP_PROTO_ICMPV6: parse_icmpv6;
            IP_PROTO_SRV6: parse_srv6;
            default: accept;
        }
    }

    ...

    state parse_srv6 {
        packet.extract(hdr.srv6h);
            transition parse_srv6_list;
    }

    state parse_srv6_list {
        packet.extract(hdr.srv6_list.next);
        bool next_segment = (bit<32>)hdr.srv6h.segment_left - 1 == (bit<32>)hdr.srv6_list.lastIndex
    ;
        transition select(next_segment) {
            true: mark_current_srv6;
            default: check_last_srv6;
        }
    }

    state mark_current_srv6 {
        local_metadata.next_srv6_sid = hdr.srv6_list.last.segment_id;
        transition check_last_srv6;
    }

    state check_last_srv6 {
        // working with bit<8> and int<32> which cannot be cast directly; using
        // bit<32> as common intermediate type for comparision
        bool last_segment = (bit<32>)hdr.srv6h.last_entry == (bit<32>)hdr.srv6_list.lastIndex;
        transition select(last_segment) {
            true: parse_srv6_next_hdr;
            false: parse_srv6_list;
        }
    }

    state parse_srv6_next_hdr {
        transition select(hdr.srv6h.next_hdr) {
            IP_PROTO_TCP: parse_tcp;
            IP_PROTO_UDP: parse_udp;
            IP_PROTO_ICMPV6: parse_icmpv6;
            default: accept;
        }
    }
}

...
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
120
121 control IngressPipeImpl (inout parsed_headers_t    hdr,
122                          inout local_metadata_t    local_metadata,
123                          inout standard_metadata_t standard_metadata) {
124
125     ...
126
127     // --- srv6_my_sid----------------------------------------------------------
128     // Process the packet if the destination IP is the segemnt Id(sid) of this
129     // device. This table will decrement the "segment left" field from the Srv6
130     // header and set destination IP address to next segment.
131
132     action srv6_end() {
133         hdr.srv6h.segment_left = hdr.srv6h.segment_left - 1;
134         hdr.ipv6.dst_addr = local_metadata.next_srv6_sid;
135     }
136
137     direct_counter(CounterType.packets_and_bytes) srv6_my_sid_table_counter;
138     table srv6_my_sid {
139         key = {
140             hdr.ipv6.dst_addr: lpm;
141         }
142         actions = {
143             srv6_end;
144         }
145
146         counters = srv6_my_sid_table_counter;
147     }
148
149     // --- srv6_transit ---------------------------------------------------------
150     // Inserts the SRv6 header to the IPv6 header of the packet based on the
151     // destination IP address.
152
153     action insert_srv6h_header(bit<8> num_segments) {
154         hdr.srv6h.setValid();
155         hdr.srv6h.next_hdr = hdr.ipv6.next_hdr;
156         hdr.srv6h.hdr_ext_len =  num_segments * 2;
157         hdr.srv6h.routing_type = 4;
158         hdr.srv6h.segment_left = num_segments - 1;
159         hdr.srv6h.last_entry = num_segments - 1;
160         hdr.srv6h.flags = 0;
161         hdr.srv6h.tag = 0;
162         hdr.ipv6.next_hdr = IP_PROTO_SRV6;
163     }
164
165     /*
166        Single segment header doesn't make sense given PSP
167        i.e. we will pop the SRv6 header when segments_left reaches 0
168      */
169     action srv6_t_insert_2(ipv6_addr_t s1, ipv6_addr_t s2) {
170         hdr.ipv6.dst_addr = s1;
171         hdr.ipv6.payload_len = hdr.ipv6.payload_len + 40;
172         insert_srv6h_header(2);
173         hdr.srv6_list[0].setValid();
174         hdr.srv6_list[0].segment_id = s2;
175         hdr.srv6_list[1].setValid();
176         hdr.srv6_list[1].segment_id = s1;
177     }
178
179     action srv6_t_insert_3(ipv6_addr_t s1, ipv6_addr_t s2, ipv6_addr_t s3) {
180         hdr.ipv6.dst_addr = s1;
181         hdr.ipv6.payload_len = hdr.ipv6.payload_len + 56;
182         insert_srv6h_header(3);
183         hdr.srv6_list[0].setValid();
184         hdr.srv6_list[0].segment_id = s3;
185         hdr.srv6_list[1].setValid();
186         hdr.srv6_list[1].segment_id = s2;
187         hdr.srv6_list[2].setValid();
188         hdr.srv6_list[2].segment_id = s1;
189     }
190
191     direct_counter(CounterType.packets_and_bytes) srv6_transit_table_counter;
192     table srv6_transit {
193         key = {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
194             hdr.ipv6.dst_addr: lpm;
195         }
196         actions = {
197             srv6_t_insert_2;
198             srv6_t_insert_3;
199         }
200         counters = srv6_transit_table_counter;
201     }
202 
203     // Called directly in the apply block.
204     action srv6_pop() {
205         hdr.ipv6.next_hdr = hdr.srv6h.next_hdr;
206         // SRv6 header is 8 bytes
207         // SRv6 list entry is 16 bytes each
208         // (((bit<16>)hdr.srv6h.last_entry + 1) * 16) + 8;
209         bit<16> srv6h_size = (((bit<16>)hdr.srv6h.last_entry + 1) << 4) + 8;
210         hdr.ipv6.payload_len = hdr.ipv6.payload_len - srv6h_size;
211 
212         hdr.srv6h.setInvalid();
213         // Need to set MAX_HOPS headers invalid
214         hdr.srv6_list[0].setInvalid();
215         hdr.srv6_list[1].setInvalid();
216         hdr.srv6_list[2].setInvalid();
217     }
218 
219     ...
220 
221     apply {
222 
223         ...
224 
225         if (do_l3_l2) {
226             // applying the routing table.
227             if (hdr.ipv6.isValid() && my_station_table.apply().hit) {
228 
229                 if (srv6_my_sid.apply().hit) {
230                     // PSP logic -- enabled for all packets
231                     if (hdr.srv6h.isValid() && hdr.srv6h.segment_left == 0) {
232                         srv6_pop();
233                     }
234                 } else {
235                     srv6_transit.apply();
236                 }
237 
238                 routing_v6_table.apply();
239                 // Check TTL, drop packet if necessary to avoid loops.
240                 if(hdr.ipv6.hop_limit == 0) { drop(); }
241             }
242 
243             // L2 bridging logic. Apply the exact table first...
244             if (!l2_exact_table.apply().hit) {
245                 // ...if an entry is NOT found, apply the ternary one in case
246                 // this is a multicast/broadcast NDP NS packet.
247                 l2_ternary_table.apply();
248             }
249         }
250 
251         // Lastly, apply the ACL table.
252         acl_table.apply();
253     }
254 }
255 
256 control DeparserImpl(packet_out packet, in parsed_headers_t hdr) {
257     apply {
258         ...
259         packet.emit(hdr.srv6h);
260         packet.emit(hdr.srv6_list);
261         ...
262     }
263 }
264 ...
```

The SRv6 header is defined (lines 15-23) as well as included the logic for parsing the header in main.p4 (lines 80-117). Next two tables are added for each of the two roles specified above (lines 138-147, 192-201).

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

In addition to the tables, the t_insert actions for policies of length 2 and 3 are implemented (lines 169-189). After this, the tables is need to be applied in the apply block at the bottom of `EgressPipeImpl` section (lines 221-254). The tables are applied after checking that the L2 destination address matches the switch's, and before the L3 table is applied because the same routing entries are used to forward traffic after the SRv6 policy is applied.

The new ONOS component that implemented here is the *Srv6Component.java*.

Srv6Component.java

```
package org.onosproject.ngsdn.tutorial;

import com.google.common.collect.Lists;
import org.onlab.packet.Ip6Address;
import org.onosproject.core.ApplicationId;
import org.onosproject.mastership.MastershipService;
import org.onosproject.net.Device;
import org.onosproject.net.DeviceId;
import org.onosproject.net.config.NetworkConfigService;
import org.onosproject.net.device.DeviceEvent;
import org.onosproject.net.device.DeviceListener;
import org.onosproject.net.device.DeviceService;
import org.onosproject.net.flow.FlowRule;
import org.onosproject.net.flow.FlowRuleOperations;
import org.onosproject.net.flow.FlowRuleService;
import org.onosproject.net.flow.criteria.PiCriterion;
import org.onosproject.net.pi.model.PiActionId;
import org.onosproject.net.pi.model.PiActionParamId;
import org.onosproject.net.pi.model.PiMatchFieldId;
import org.onosproject.net.pi.model.PiTableId;
import org.onosproject.net.pi.runtime.PiAction;
import org.onosproject.net.pi.runtime.PiActionParam;
import org.onosproject.net.pi.runtime.PiTableAction;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.onosproject.ngsdn.tutorial.common.FabricDeviceConfig;
import org.onosproject.ngsdn.tutorial.common.Utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.List;
import java.util.Optional;

import static com.google.common.collect.Streams.stream;
import static org.onosproject.ngsdn.tutorial.AppConstants.INITIAL_SETUP_DELAY;

/**
 * Application which handles SRv6 segment routing.
 */
@Component(
        immediate = true,
        enabled = true,
        service = Srv6Component.class
)
public class Srv6Component {

    private static final Logger log = LoggerFactory.getLogger(Srv6Component.class);

    //--------------------------------------------------------------------------
    // ONOS CORE SERVICE BINDING
    //
    // These variables are set by the Karaf runtime environment before calling
    // the activate() method.
    //--------------------------------------------------------------------------

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private FlowRuleService flowRuleService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private MastershipService mastershipService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
66     private DeviceService deviceService;
67
68     @Reference(cardinality = ReferenceCardinality.MANDATORY)
69     private NetworkConfigService networkConfigService;
70
71     @Reference(cardinality = ReferenceCardinality.MANDATORY)
72     private MainComponent mainComponent;
73
74     private final DeviceListener deviceListener = new Srv6Component.InternalDeviceListener();
75
76     private ApplicationId appId;
77
78     //--------------------------------------------------------------------------
79     // COMPONENT ACTIVATION.
80     //
81     // When loading/unloading the app the Karaf runtime environment will call
82     // activate()/deactivate().
83     //--------------------------------------------------------------------------
84
85     @Activate
86     protected void activate() {
87         appId = mainComponent.getAppId();
88
89         // Register listeners to be informed about device and host events.
90         deviceService.addListener(deviceListener);
91
92         // Schedule set up for all devices.
93         mainComponent.scheduleTask(this::setUpAllDevices, INITIAL_SETUP_DELAY);
94
95         log.info("Started");
96     }
97
98     @Deactivate
99     protected void deactivate() {
100        deviceService.removeListener(deviceListener);
101
102        log.info("Stopped");
103    }
104
105    /**
106     * Populate the My SID table from the network configuration for the
107     * specified device.
108     *
109     * @param deviceId the device Id
110     */
111    private void setUpMySidTable(DeviceId deviceId) {
112
113        Ip6Address mySid = getMySid(deviceId);
114
115        log.info("Adding mySid rule on {} (sid {})...", deviceId, mySid);
116
117
118        // Fill in the table ID for the SRv6 my segment identifier table
119        String tableId = "IngressPipeImpl.srv6_my_sid";
120
121        // Modify the field and action id to match your P4Info
122        PiCriterion match = PiCriterion.builder()
123                .matchLpm(
124                        PiMatchFieldId.of("hdr.ipv6.dst_addr"),
125                        mySid.toOctets(), 128)
126                .build();
127
128        PiTableAction action = PiAction.builder()
129                .withId(PiActionId.of("IngressPipeImpl.srv6_end"))
130                .build();
131
132        FlowRule myStationRule = Utils.buildFlowRule(
133                deviceId, appId, tableId, match, action);
134
135        flowRuleService.applyFlowRules(myStationRule);
136    }
137
138    /**
139     * Insert a SRv6 transit insert policy that will inject an SRv6 header for
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
140      * packets destined to destIp.
141      *
142      * @param deviceId     device ID
143      * @param destIp       target IP address for the SRv6 policy
144      * @param prefixLength prefix length for the target IP
145      * @param segmentList  list of SRv6 SIDs that make up the path
146      */
147     public void insertSrv6InsertRule(DeviceId deviceId, Ip6Address destIp, int prefixLength,
148                                      List<Ip6Address> segmentList) {
149         if (segmentList.size() < 2 || segmentList.size() > 3) {
150             throw new RuntimeException("List of " + segmentList.size() + " segments is not
    supported");
151         }
152
153
154         // Fill in the table ID for the SRv6 transit table.
155         String tableId = "IngressPipeImpl.srv6_transit";
156
157         // Modify match field, action id, and action parameters to match your P4Info.
158         PiCriterion match = PiCriterion.builder()
159                 .matchLpm(PiMatchFieldId.of("hdr.ipv6.dst_addr"), destIp.toOctets(), prefixLength)
160                 .build();
161
162         List<PiActionParam> actionParams = Lists.newArrayList();
163
164         for (int i = 0; i < segmentList.size(); i++) {
165             PiActionParamId paramId = PiActionParamId.of("s" + (i + 1));
166             PiActionParam param = new PiActionParam(paramId, segmentList.get(i).toOctets());
167             actionParams.add(param);
168         }
169
170         PiAction action = PiAction.builder()
171                 .withId(PiActionId.of("IngressPipeImpl.srv6_t_insert_" + segmentList.size()))
172                 .withParameters(actionParams)
173                 .build();
174
175         final FlowRule rule = Utils.buildFlowRule(
176                 deviceId, appId, tableId, match, action);
177
178         flowRuleService.applyFlowRules(rule);
179     }
180
181     /**
182      * Remove all SRv6 transit insert polices for the specified device.
183      *
184      * @param deviceId device ID
185      */
186     public void clearSrv6InsertRules(DeviceId deviceId) {
187         // Fill in the table ID for the SRv6 transit table
188         String tableId = "IngressPipeImpl.srv6_transit";
189
190         FlowRuleOperations.Builder ops = FlowRuleOperations.builder();
191         stream(flowRuleService.getFlowEntries(deviceId))
192                 .filter(fe -> fe.appId() == appId.id())
193                 .filter(fe -> fe.table().equals(PiTableId.of(tableId)))
194                 .forEach(ops::remove);
195         flowRuleService.apply(ops.build());
196     }
197
198     // ---------- END METHODS TO COMPLETE ----------------
199
200     //------------------------------------------------------------------------------
201     // EVENT LISTENERS
202     //
203     // Events are processed only if isRelevant() returns true.
204     //------------------------------------------------------------------------------
205
206     /**
207      * Listener of device events.
208      */
209     public class InternalDeviceListener implements DeviceListener {
210
211         @Override
212         public boolean isRelevant(DeviceEvent event) {
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

```
213             switch (event.type()) {
214                 case DEVICE_ADDED:
215                 case DEVICE_AVAILABILITY_CHANGED:
216                     break;
217                 default:
218                     // Ignore other events.
219                     return false;
220             }
221             // Process only if this controller instance is the master.
222             final DeviceId deviceId = event.subject().id();
223             return mastershipService.isLocalMaster(deviceId);
224         }
225 
226         @Override
227         public void event(DeviceEvent event) {
228             final DeviceId deviceId = event.subject().id();
229             if (deviceService.isAvailable(deviceId)) {
230                 // A P4Runtime device is considered available in ONOS when there
231                 // is a StreamChannel session open and the pipeline
232                 // configuration has been set.
233                 mainComponent.getExecutorService().execute(() -> {
234                     log.info("{} event! deviceId={}", event.type(), deviceId);
235 
236                     setUpMySidTable(event.subject().id());
237                 });
238             }
239         }
240     }
241 
242 
243     //------------------------------------------------------------------------------
244     // UTILITY METHODS
245     //------------------------------------------------------------------------------
246 
247     /**
248      * Sets up SRv6 My SID table on all devices known by ONOS and for which this
249      * ONOS node instance is currently master.
250      */
251     private synchronized void setUpAllDevices() {
252         // Set up host routes
253         stream(deviceService.getAvailableDevices())
254                 .map(Device::id)
255                 .filter(mastershipService::isLocalMaster)
256                 .forEach(deviceId -> {
257                     log.info("*** SRV6 - Starting initial set up for {}...", deviceId);
258                     this.setUpMySidTable(deviceId);
259                 });
260     }
261 
262     /**
263      * Returns the Srv6 config for the given device.
264      *
265      * @param deviceId the device ID
266      * @return Srv6  device config
267      */
268     private Optional<FabricDeviceConfig> getDeviceConfig(DeviceId deviceId) {
269         FabricDeviceConfig config = networkConfigService.getConfig(deviceId, FabricDeviceConfig.
    class);
270         return Optional.ofNullable(config);
271     }
272 
273     /**
274      * Returns Srv6 SID for the given device.
275      *
276      * @param deviceId the device ID
277      * @return SID for the device
278      */
279     private Ip6Address getMySid(DeviceId deviceId) {
280         return getDeviceConfig(deviceId)
281                 .map(FabricDeviceConfig::mySid)
282                 .orElseThrow(() -> new RuntimeException(
283                         "Missing mySid config for " + deviceId));
284     }
285 }
```


Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

To show that traffic can be steered using an SRv6 policy a ping is started between hosts h2 and h4.

```
mininet> h2 ping h4
```

Using the ONOS UI, can be observed which paths are being used for the ping packets, Figure 29.

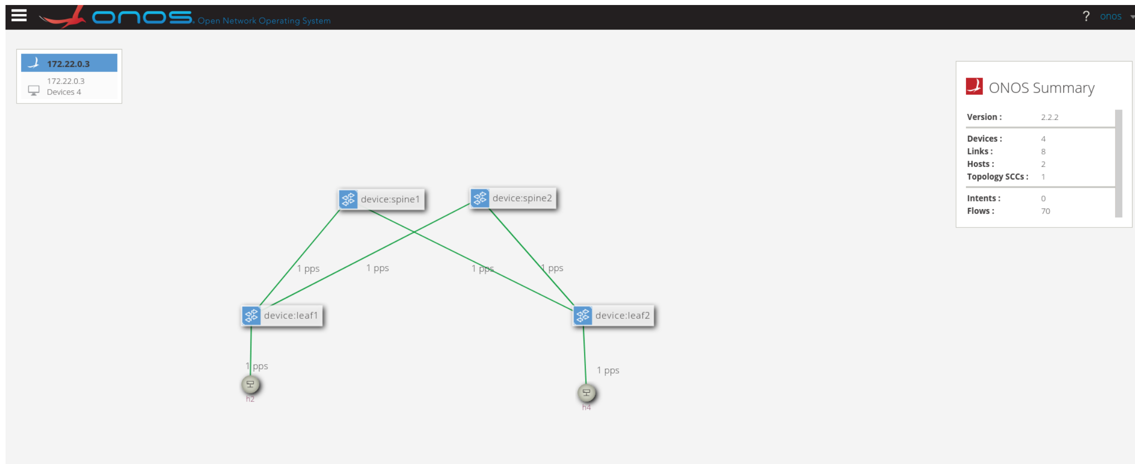


Figure 29: ONOS UI SRv6 policy not used

To insert a set of SRv6 policies that sends the ping packets via the other spine:

```
onos> srv6-insert &lt;device ID&gt; &lt;segment list&gt;
```

To add a policy that forwards traffic between h2 and h4 though spine1 and leaf2, you can use the following command:

- Insert the SRv6 policy from h2 to h4 on leaf1 (through spine1 and leaf2)

```
onos> srv6-insert device:leaf1 3:201:2:: 3:102:2:: 2001:1:4::1
Installing path on device device:leaf1: 3:201:2::, 3:102:2::, 2001:1:4::1
```

- Insert the SRv6 policy from h4 to h2 on leaf2 (through spine1 and leaf1)

```
onos> srv6-insert device:leaf2 3:201:2:: 3:101:2:: 2001:1:2::1
Installing path on device device:leaf2: 3:201:2::, 3:101:2::, 2001:1:2::1
```

In this topology, the SID for spine1 is 3:201:2:: and the SID for spine is 3:202:2::.
To confirm that the rule has been added using a variant of the following:

```
onos> flows any device:leaf1 | grep tableId=IngressPipeImpl.srv6_transit
    id=c000006d73f05e, state=ADDED, bytes=0, packets=0, duration=871, liveType=UNKNOWN, priority
    =10,
    tableId=IngressPipeImpl.srv6_transit,
    appId=org.p4.srv6-tutorial,
    selector=[hdr.ipv6.dst_addr=0x20010001000400000000000000000001/128],
    treatment=DefaultTrafficTreatment{immediate=[
        IngressPipeImpl.srv6_t_insert_3(
            s3=0x20010001000400000000000000000001,
            s1=0x30201000200000000000000000000,
            s2=0x30102000200000000000000000000)],
    deferred=[], transition=None, meter=[], cleared=false, StatTrigger=null, metadata=null}
```

Using the ONOS UI is confirmed that traffic is flowing through the specified spine. Figure 30 shows the case.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

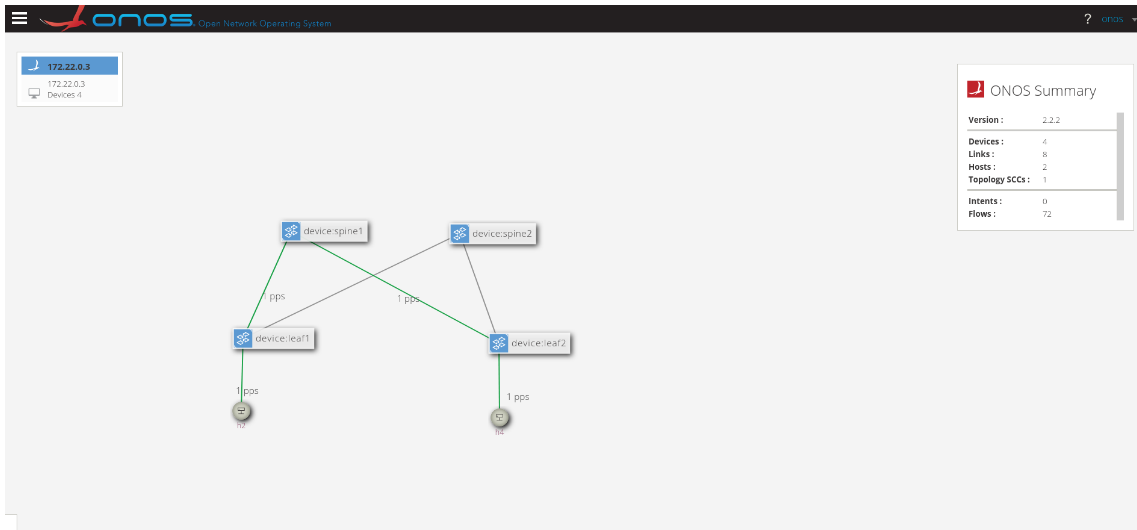


Figure 30: ONOS UI SRv6 policy used

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

7 Conclusion and Future Work

In this thesis, some of the existing solutions used in today's business networks, such as SDN and NFV, as well as the benefits and the unsolved problems of these technologies are presented. Furthermore, a new concept concerning the data plane programming using P4 programming language was introduced and its advantages was pointed out. In addition, several showcases regarding the integration of ONOS an SDN controller and P4 were presented, which can provide business with a better approach when designing their network. To that end, some simple protocols, such as IPv6 Routing with ECMP and Segement Routing v6 were implemented. The results of this thesis are of great value.

Future research plans include applying a layer-2 handover management functionality, to create a programmable data plane and ONOS SDN controller for a programmable control plane; taking advantage of the remarkable advantages of P4 programming, it is expected that binding update latency and traffic towards the network controller can be minimized.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

References

[1] Nick McKeown et al. ≪OpenFlow: Enabling Innovation in Campus Networks≫. In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69{74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. URL: https://doi.org/10.1145/1355734.1355746.

[2] Raphael Horvath, Dietmar Nedbal, and Mark Stieninger. ≪A Literature Review on Challenges and Effects of Software Defined Networking≫. In: *Procedia Computer Science* 64 (2015). Conference on ENTERprise Information Systems/International Conference on Project MANagement/Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2015 October 7-9, 2015, pp. 552{561. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.08.563. URL: http://www.sciencedirect.com/science/article/pii/S1877050915026988.

[3] W. Xia et al. ≪A Survey on Software-Defined Networking≫. In: *IEEE Communications Surveys Tutorials* 17.1 (2015), pp. 27{51. DOI: 10.1109/COMST.2014.2330903.

[4] Rahim Masoudi and Ali Ghaffari. ≪Software defined networks: A survey≫. In: *Journal of Network and Computer Applications* 67 (2016), pp. 1{25. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.03.016. URL: http://www.sciencedirect.com/science/article/pii/S1084804516300297.

[5] Sahrish Khan Tayyaba et al. ≪Software-Defined Networks (SDNs) and Internet of Things (IoTs): A Qualitative Prediction for 2020≫. In: *International Journal of Advanced Computer Science and Applications* 7.11 (2016). DOI: 10.14569/IJACSA.2016.071151. URL: http://dx.doi.org/10.14569/IJACSA.2016.071151.

[6] Péter Voros and Attila Kiss. ≪Security Middleware Programming Using P4≫. In: *Human Aspects of Information Security, Privacy, and Trust*. Ed. by Theo Tryfonas. Cham: Springer International Publishing, 2016, pp. 277{287. ISBN: 978-3-319-39381-0.

[7] Sumit Badotra. ≪A Review Paper on Software Defined Networking≫. In: *International Journal of Advanced Computer Research* 8 (Mar. 2017).

[8] Henning Stubbe. ≪P4 Compiler and Interpreter: A Survey≫. In: *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM)*. Ed. by Georg Carle, Daniel Raumer, and Lukas Schwaighofer. Chair of Network Architectures and Services, 2017, pp. 47{52.

[9] P. Prabakaran, D. P. Isravel, and S. Silas. ≪A Review of SDN-Based Next Generation Smart Networks≫. In: *2019 3rd International Conference on Computing and Communications Technologies (ICCCT)*. 2019, pp. 80{85.

[10] Deepak Rana, Shiv Dhondiyal, and Sushil Chamoli. ≪Software Defined Networking (SDN) Challenges, issues and Solution≫. In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* 7 (Jan. 2019), pp. 884{889. DOI: 10.26438/ijcse/v7i1.884889.

[11] Ricard Vilalta et al. ≪Network Programmability and Automation in Optical Networks≫. In: *Optical Network Design and Modeling*. Ed. by Anna Tzanakaki et al. Cham: Springer International Publishing, 2020, pp. 223{234. ISBN: 978-3-030-38085-4.

[12] URL: https://plvision.eu/rd-lab/blog/sdn/p4-programming-future-sdn.

[13] URL: https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture/.

[14] URL: https://www.routerfreak.com/software-defined-network-use-cases-from-the-real-world/.

[15] URL: https://www.sdxcentral.com/networking/sdn/definitions/what-is-sdn-controller/.

[16] URL: https://www.sdxcentral.com/networking/nfv/definitions/whats-network-functions-virtualization-nfv/.

[17] URL: https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html.

[18] URL: https://opennetworking.org/onos/.

[19] URL: https://programmersought.com/article/76604747557/.

[20] URL: https://wiki.onosproject.org/.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105

[21] URL: https://wiki.opennetworking.org/plugins/viewsource/viewpagesrc.action?pageId=309231646.

[22] URL: https://www.openconfig.net/.

[23] URL: https://github.com/Yi-Tseng/Yi-s-gNMI-tool.

[24] Internet Engineering Task Force (IETF). *Neighbor Discovery for IP version 6 (IPv6)*. URL: https://tools.ietf.org/html/rfc4861.

[25] Internet Engineering Task Force (IETF). *Network Configuration Protocol (NETCONF)*. URL: https://tools.ietf.org/html/rfc6241.

[26] Internet Engineering Task Force (IETF). *RESTCONF Protocol*. URL: https://tools.ietf.org/html/rfc8040.

[27] Internet Engineering Task Force (IETF). *SRv6 Network Programming*. URL: https://tools.ietf.org/id/draft-filsfils-spring-srv6-network-programming-06.html.

[28] Internet Engineering Task Force (IETF). *The YANG 1.1 Data Modeling Language*. URL: https://tools.ietf.org/html/rfc7950.

[29] Internet Engineering Task Force (IETF). *Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels*. URL: https://tools.ietf.org/html/rfc6438.

[30] Internet Engineering Task Force (IETF). *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. URL: https://tools.ietf.org/html/rfc6020.

[31] A.Roussel et al. *nanomsg*. URL: http://nanomsg.org/.

[32] P4 Language Consortium. *Behavioral Model (bmv2)*. URL: https://github.com/p4lang/behavioral-model.

[33] P4 Language Consortium. *P4 Language Specification*. URL: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf.

[34] P4 Language Consortium. *p4c-behavioral*. URL: https://github.com/p4lang/p4c-behavioral.

[35] P4 Language Consortium. *p4c-hlir*. URL: https://github.com/p4lang/p4-hlir.

[36] P4 Language Consortium. *P4Runtime Shell*. URL: https://github.com/p4lang/p4runtime-shell.

[37] P4 Language Consortium. *Preprocessor for the P4behavioral model*. URL: https://github.com/p4lang/p4c-bm.

[38] P4 Language Consortium. *switch*. URL: https://github.com/p4lang/switch.

[39] *Docker*. URL: https://www.docker.com/.

[40] *ETSI*. URL: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/.

[41] *Floodlight Loxigen*. URL: https://github.com/floodlight/loxigen/wiki/OpenFlowJ-Loxi.

[42] Apache Software Foundation. *Apache Thrift*. URL: https://thrift.apache.org/.

[43] Google. *Protocol Buffers - Google's data interchange format*. URL: https://github.com/protocolbuffers/protobuf.

[44] Open Networking Laborator. *Mininet*. URL: http://mininet.org/.

[45] A. Metzler and Associates. *Ten Things to Look for in an SDN Controller*. URL: http://www.webtorials.com/main/resource/papers/webtorials/Metzler/paper1/10_Factors_SDN_Controller.pdf.

[46] Cumulus Networks. *Open Network Install Environment*. URL: http://onie.org/.

[47] OpenConfig. *gNMI - gRPC Network Management Interface*. URL: https://github.com/openconfig/gnmi.

[48] J. Pan. *Software Reliability, Carnegie Mellon University*.

[49] Open Compute Project. *Switch Abstraction Interface*. URL: https://github.com/opencomputeproject/SAI.

[50] *Virtual Network Function*. URL: https://www.sdxcentral.com/networking/nfv/definitions/virtual-network-function/.

Institutional Repository - Library & Information Centre - University of Thessaly
20/05/2024 02:05:20 EEST - 18.191.234.105