DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2016

Scalable 5-Tuple Packet Classification in Overlay Network-Based SDN

MUHAMMAD ARIF

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING

Master Thesis Report

Scalable 5-Tuple Packet Classification in Overlay Network-Based SDN

Author:
Muhammad Arif

Advisor:
Dr. Albert Cabellos
Universitat Politècnica de Catalunya (UPC)

Dr. Viktoria Fodor
Kungliga Tekniska Högskolan (KTH)

Presented on:
31 August 2016

Abstract

Traditional networking paradigm, with destination-based forwarding, provides low processing latency in terms of lookup time and it can scale to huge number of rules for traffic engineering. On the other hand, it lacks flexibility in terms of the packet header fields that can be used to implement the traffic engineering rules. The global view of the network or network abstraction is also lacking here, hence it is harder to program the network.

The flexibility and programmability problem are two reasons, among others, that motivate the rise of the Software-Defined Network (SDN) paradigm: higher flexibility for traffic engineering and easier programmability for finer traffic engineering rules. SDN offers a huge degree of flexibility in the network for traffic engineering, for example in an OpenFlow-based network infrastructure, up to 38 packet header fields can be observed. This degree of flexibility comes however with a cost in terms of lookup times for packet forwarding, it does not scale for huge number of rules. The SDN paradigm also introduces the centralized control plane, which makes it easier to program the network. One of the well-known implementation of SDN is an overlay network, which builds a virtual network as an abstraction of the physical network. It simplifies the logical topology and provides more flexibility in terms of network programmability. In the overlay network implementation the rules are placed centrally in the centralized controller, rather than distributed to the network devices, hence it is easier to manage and program the network. The drawback of the centralized rules storing is that the requirement of storage space to store the rules is increasing significantly.

Consequently, while SDN offers high flexibility and network programmability, it comes with problems for traffic engineering: high processing latency and storage requirement. With the increasing number of applications hosted in the network and the increasing needs for finer traffic engineering, more scalable ways to implement finer traffic engineering rules are needed, so the system can scale even with high number of rules.

In this thesis, we address of problem of rule aggregation, a process to combine multiple rules without losing the accuracy of the original individual rules. We also address the problem of packet classification, a process to decide which flow that a packet belongs to and to determine which action needs to be taken for that packet. We propose one possible solution for rule aggregation and packet classification for overlay networks, focusing on 5-tuple traffic engineering rules, with the goal to minimize the storage space requirement and processing latency. The observed system performance metrics are the number of entries stored in the system, the number of entries observed for classification and the lookup times. The proposed solution is evaluated by means of system-level simulation and implementation in the Open Overlay Router and Vector Packet Processing (VPP) platform using synthetic rules generated with real-world distribution. The results are compared with a system without using the proposed rule aggregation and packet classification method. The results show that with the incorporation of both methods, the requirement for storage space and the processing latency can be reduced significantly. As an example, we note that 58.6% maximum saving in the storage required and 29.6% maximum delay reduction can be achieved.

Abstrakt

Den traditionella nätverk paradigm, med destination baserad vidarebefordring, ger låg bearbetning fördröjning i form av lookup tider och det kan skalas till stort antal regler för trafikstyrning. Å andra sidan, det saknar flexibilitet när det gäller paketrubrikfält, som kan användas för att genomföra trafikstyrning reglerna. Den globala utsiktet över nätverket eller nätverksabstraktion saknas också här, och därför är det svårare att programmera nätverket.

Problem med flexibiliteten och programmerbarheten problem är två skäl, som, bland annat, motiverar ökningen av Programvarustyrd Network (SDN) paradigm: större flexibilitet för trafikstyrning och enklare programmerbarhet för finare trafikstyrningsreglerna. SDN erbjuder ett stort mått av flexibilitet i nätverket för trafikstyrning, till exempel i en Openflow-baserad nätverksinfrastruktur, upp till 38 pakethuvudfält kan observeras. Denna flexibilitet kommer dock med en kostnad i form av lookup tider för paketbefordran, det skalar inte för stort antal regler. SDN paradigmet inför också den centraliserade styrplanet, vilket gör det lättare att programmera nätverket. En av de välkända genomförande av SDN är ett overlay nät, som bygger ett virtuellt nätverk som en abstraktion av det fysiska nätverket. Det förenklar den logiska topologin och ger mer flexibilitet när det gäller nätverksprogrammerbarhet. I overlay nätet är reglerna placerade i centraliserade kontrollern, snarare än distribuerade till nätverksenheterna, därför är det lättare att hantera och programmera nätverket. Nackdelen med den centraliserade lagring av reglerna är att kravet på lagringsutrymme för att lagra reglerna ökar avsevärt.

Följaktligen medan SDN ger hög flexibilitet och nätverksprogrammerbarhet, kommer det med problem för trafikstyrning: hög processorfördröjning och krav för lagring. Med det ökande antalet applikationer i nätverket och de ökande behoven av finare trafikstyrning, behövs det mer skalbara sätt att genomföra finare trafikstyrningsregler, så att systemet kan skala även med stort antal regler.

I denna avhandling tar vi itu med problemet med aggregering av reglarna, en process att kombinera flera regler utan att förlora noggrannheten av de ursprungliga enskilda reglerna. Vi vänder oss också till problemet med paketklassificering, en process att avgöra vilket flöde som ett paket tillhör och för att avgöra vilka åtgärder som behöver vidtas för att viderbefordra paketet. Vi föreslår en möjlig lösning för regelnaggregering och paketklassificering för overlay-nätverk, med fokus på 5-tuple trafikstyrning regler, med målet att minimera lagerutrymmeskrav och bearbetning latens.

De observerade prestandastatistik är antalet regler som lagrats i systemet, antalet regler som observerades för klassificering och lookup fördröjning.

Den föreslagna lösningen utvärderas med hjälp av systemnivåsimulering och implementering i en Open Overlay Router och Vector Packet Processing (VPP) plattform med hjälp av syntetiska regler som genereras med verklig fördelning. Resultaten jämförs med ett system utan att använda den föreslagna aggregering och paketklassificeringsmetod. Resultaten visar att med införandet av båda metoderna kan kravet på lagringsutrymme och bearbetningsfördröjning minskas avsevärt. Som ett exempel kan 58,6% maximal besparing i lagringsutrymme och 29,6% maximal fördröjning reduktion uppnås.

Acknowledgement

I would like to express my gratitude to Dr. Albert Cabellos, for his continouis support and guidance during the thesis work. I would also like to credit Alberto Rodriguez-Natal, Albert Lopez and the team from UPC, Barcelona, and also Florin Coras from Cisco Systems for all the helps provided throughout this thesis work.

I'm also very grateful to Dr. Viktoria Fodor as my advisor and examiner at KTH, for all the helps, feedbacks and advices since the start of this thesis work.

Finally, I would also like express my gratitude to my friends and family, especially my wife, for all the support throughout my master study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, Internet has been an integral part of our life and regarded as indispensable in this era. Internet is basically interconnectivity of computer networks with Internet Protocol suite (TCP/IP) as the communication protocol. The underlying architecture of Internet has not change much since it was designed many years ago, with IP as the key network protocol. Traditionally, Internet offers best-efforts service with IP destination-based forwarding. This method has been proven to provide low processing latency, in terms lookup time, for packet classification, a process to find which flow that a packet belongs to and determine which action needs to be taken for the packet. For the traditional networking, it only considers single field (destination address) for packet classification, hence the required space to store each rule is small. Because of that, it is known that traditional networking can scale to huge number of rules for traffic engineering.

With the tremendous growth of internet utilization, many applications are hosted in the Internet, such as websites, e-mail, video streaming, etc. Because of that, instead of best-efforts, the future IP networks will need to offer enhanced services to the users [30], with finer flexibility to implement traffic engineering. For example, to set maximum allocated bandwidth for video streaming applications, HTTP access needs to be routed via specific network for content filtering or the access to specific application needs to go through firewall for security reason. The traditional networking paradigm, with destination-based forwarding, is lacking of that flexibility, since the detection of those applications cannot be done by only observing the destination address, but also port number, protocol and/or other fields in the packet header.

On top of that, in the traditional networking paradigm, each network device has its own intelligence to make forwarding decision, so the traffic engineering rules need to be implemented in each network device. There is no global view or abstraction of the network. This makes it more complex to program the network, in order to implement finer traffic engineering rules. The lack of flexibility and complexity to program the network for finer traffic engineering rules of the current traditional networking paradigm were two reasons, among others, that motivate the rise of Software-Defined Networking (SDN) paradigm.

SDN paradigm offers a huge degree of flexibility of the network. The implementation of SDN, such as OpenFlow-based SDN switches, allows more flexibility in terms of packet header fields to observe for packet classification. As much as 38 packet header fields can be observed in OpenFlow-based switch [31]. SDN is also easier to program for implementing finer traffic engineering rules. It decouples the control plane from the data plane , where the forwarding decision is made, capability from the network devices and build logically centralized control plane. By doing so, the control plane now has the global view of the network, so it is easier to implement complex traffic engineering rules. One of the well-known implementation of SDN is overlay network, where a virtual network is built on top of the physical network, to provide simpler logical topology of the network, hence it is easier to program.

With the increasing number of application hosted in the network, the number of rules for traffic engineering is increasing significantly. On top of that, the requirement for more application-specific traffic engineering is also increasing. As shown in the examples above, the application-specific traffic engineering rules needs to observe more packet header fields. Because of those reasons, the required space to store more rules, with more fields in each rules, is increasing, along with the increase of lookup time when a packet needs to be classified. The introduction of SDN gives flexibility and centralization of the control plane that provides the capabilities to observe more packet header fields and make the network easier to program, even for complex rules. On the other hand, those capabilities does not scale well for huge number of rules, since the processing latency and requirement for storage to store the rules increases. Those problems are resulting in the need of more scalable packet classification with many fields to match. Scalable packet classification has two main goals: faster lookup time for packet classification and rule aggregation. Rule aggregation is a process of combining multiple traffic engineering rules without losing accuracy of the original individual rules, in order to reduce the storage space requirement.

Scalable packet classification with many-field has gained massive interest from both industry and academic world [7, 8]. There are some proposed solution, with their own pros and cons. Rottenstreich, et al, in [6], discussed about the implementation of many-field packet classification in TCAM (Ternary Content Addressable Memory), but it comes with high implementation cost due to the increase rule size (more fields to be stored). There were some papers, such as [12, 13], that discussed about implementing many-field classification using decision-tree, but when the number of fields in a rule is increasing, the memory requirement becomes very high and the size of the tree is increasing in both axis, hence the processing latency is also increasing.

1.1 Problem Statement

From our description above, we can see that the increasing number of application hosted in the network comes with problems for packet classification, in terms of lookup time and the storage space requirement. The network owners, therefore, needs to consider the trade-off between more flexible traffic engineering and high performance packet classification. As we can see from the previous works above, flexibility in terms of the fields matched in the classification resulted in the increasing requirement of storage and/or memory usage and also the increase of processing latency. On the other hand, based on our observation, only minority of network owners are using the whole 38 packet headers field provided by OpenFlow. This provides the possibility to find a solution where some degree of flexibility can be achieved while minimizing the performance impact.

In this thesis, we propose rule aggregation and flow classification method to reduce both space requirement and processing latency, while also providing some degree of flexibility in terms of the packet header fields to be observed. The flexibility that we provide in this method is classification with 5-tuple information, including source/destination address, port numbers and protocol. We do believe that for majority number of network owners, those information fields are adequate. We also implement the proposed methods in the overlay network scenario with Open Overlay Router (OOR) as the control plane and Vector Packet Processing (VPP) as the data plane. Lastly, we evaluate the performance of the proposed methods in different set of synthetic rules generated with real-world distribution.

Our first objective is to propose rule aggregation algotrithm to minimize the storage space requirements for a given rulesets with various real-world distribution in a given network, while still maintaining the flexibility specified above. In this way, we can analyze the maximum achievable reduction number of rules by incorporating the aggregation algorithm compared to original ruleset without aggregation.

Our second objective is to propose packet classification method to minimize the processing latency for the packet classification in terms of lookup time with the same setup as previous objective, while still maintaining the accuracy of the packet classification. In this way, we can analyze the maximum achievable reduction of lookup time by incorporating the classification method compared to current classification method.

1.2 Related Works

In this section, several previous studies that relevant to the thesis are summarized, mainly in the area of packet classification. Packet classification has been a well-known problem that is interesting for the industry and academic world. There are many proprosed methods for packet classification with their own pros and cons.

In the past, decision-tree-based approach for packet classification, such as HiCuts [12] and HyperCuts [9], has been proven to give a good packet classification performance by constructing decision tree of the packet header fields information. The decision-tree is built with each packet header fields as the branch node. Incoming packets are checking each branch until a leaf is found. The size of the tree, especially the number of branches, relates directly to the number of packet header fields that needs to be observed during the classification process. Lim, et al, in [13] proposed improvement for decision-tree based approach, by using boundary cutting, to increase the effectiveness of branching the decision-tree. The increasing number of packet header fields in the rules and the rule size for finer traffic engineering is causing the size of the tree to grow significantly, hence the requirement for storage also increases significantly, along with the increase of packet classification latency.

Rottenstreich, et al, in [6], explored the implementation of many field packet classification by leveraging TCAM (Ternary Content Addressable Memory), a specialized memory for high-speed lookup that can accomodate binary entries and wildcards. In the paper, they presented an algorithm that analyze optimal encoding to represent typical real-life classification database. They also proposed a solution by combining regular TCAM, to store simple rules, and modified TCAM, to store more complex rules. Other solution that leverages TCAM for packet classification was proposed in[34], it combined the usage of TCAM and decision-tree. SAX-PAC [8] is also using TCAM for packet classification and leveraging the capabilities of paralel computing to do fast packet classification. Those proposed solutions give a low latency for packet classification. On the other hand, with the increasing number of rules and rule sizes, more TCAM devices are needed, hence it leads to high implementation cost.

One other well-known packet classification method is using Tuple Space Search solution. Srinivasan, et al, in [4] proposed this solution, an algorithmic approach for software-based packet classification. In this approach, they proposed a solution to examine the tuples (packet header fields) in the database and combine the rules with same tuple into a single hash table. If new rule is added and it has different tuple with the available hash tables, new hash table will be created. Incoming packets need to checks all tables, even after a match is found. In the case of two or more matches are found, the match with highest priority will be used. This approach has been used in many implementation, and one well-known implementation is in Open vSwitch (OVS) [33]. On the other hand, with the increasing number of tuples that needs to be observed to implement finer traffic engineering, the number of hash tables are also increasing. It can lead to higher processing latency, especially with more number of rules in the database.

Recently, Hsieh, et al, proposed a solution for packet classification with multidimensional-cutting via selective bit concatenation (MC-SBC) [20]. For this solution, they introduced the concept of pre-filtering by selecting certain bit position to split the rules in the database to multiple lookup tables with smaller number of rules in each lookup table. This solution is different with with Tuple Space Search, that uses the entire tuple to split the database. On top of that, rather than letting the incoming packets to check all tables, MC-SBC is also doing a pre-filtering process for the packets and directing the packet to a specific lookup tables.

In this thesis, we propose a solution for scalable packet classification, to reduce the processing latency, along with rule aggregation, to reduce the storage space requirement. The ideas from [20, 4] are used as the basic of the packet classification method proposal, along with the introduction of rule aggregation algorithm. In [20], the original rules were used for the calculation of to decide which bit positions used to split the rules into multiple tables, but in our proposal, the aggregated rules are used for to do the same. Other modification that we introduce in the proposed methods in this thesis is that we remove the requirement of calculating wildcard ratio to decide the bit position for pre-filtering. Our rule aggregation algorithm assures that wildcards only appear in the predicted position, hence before the calculation, any bit positions with wildcards can be removed from the candidate bit position for pre-filtering. By doing so, other than reducing the calculation complexity, it also ensures that each can only be placed in a single lookup table. This modification results in the decrease of processing latency and storage requirement, especially when many rules with wildcards are introduced by the users.

1.3 Methodology

This thesis presents an approach to tackle the problem of many-field packet classification in SDN and overlay network with experimental research methodology. Experimental research methodology is typical research methodology in computer networks. By choosing this approach, we aim to demonstrate and evaluate the performance of the proposed solution by building prototype of it.

The process in this thesis can be broken down into four main phases: preliminary study, development, implementation and evaluation. In the beginning of this thesis, preliminary study is conducted, which can be divided into literature review and implementation and code review of the current systems. Literature review was conducted to get better understanding of the current state of the art technology in SDN, overlay network and many-field packet classification. Implementation and code review of the existing systems was also done to understand better about how Open Overlay Router (OOR) and Vector Packet Processing (VPP) platforms works. Then, we developed our solution for the overlay network based on the knowledge accumulated from the preliminary study. The solution that we developed was rule aggregation algorithm, based on the details of how VPP classifier works, and flow classification method, based on multidimensional-cutting via selective bit-concatenation method. The next phase was the implementation phase, where we built the prototype of the proposed solution. We then evaluated the performance of the prototype by integrating it to the aforementioned platforms and testing it in the testbed that simulate real-world simple network environment conditions.

1.4 Report Structure

The rest of the report is organized as follows. In Chapter 2, we discuss about the key terms: SDN, LISP and Overlay Network. Chapter 3 and 4 is dedicated to explain the platform used for the implementation and testing, Open Overlay Router and Vector Packet Processing. Then we discuss the detail of the development process and the proposed methods in Chapter 5. The results of the simulation and testing is presented in Chapter 6. In Chapter 7, we include our analysis, conclusion and our suggestion for future researches.

Chapter 2

Background

In this chapter, we review the basic technologies used in this thesis. First, we discuss about SDN, as the paradigm that we use in the thesis, and its advantages. Then, we discuss about overlay network, as the implementation of SDN that we use in this thesis, the basic concept of it and its advantages. By implementing overlay network, we can have the abstraction of the network and easiness to program the network, without the need to make any change in the current network. One of the available implementation for overlay network that we use in this thesis is Open Overlay Router (OOR), an open-source platform to create programmable overlay network. OOR uses Locator/ID Separation Protocol (LISP) as the base of their architecture. We discuss about the basic of LISP and its components in the last part of this section, to understand better about OOR, that will be discussed in the next chapter.

2.1 Software-Defined Networking (SDN)

SDN is the current trend in the networking world, it introduces a new paradigm which changes the network architecture. Based on the definition from Open Networking Foundation (ONF)[3], SDN decouples the control plane of a network from the data plane or the forwarding plane and it is directly programable. Control plane is where the forwarding decision is made and data plane is where the actual packet forwarding process takes place by following the decision made by control plane.

In the traditional network, both planes are tied together, hence networking devices has their own intellegence to handle incoming and outgoing packets or traffic. Due to this distributed intelligence, the devices do not have a global view of the network, hence it is more complex to configure finer traffic engineering for big network. SDN offers a solution to tackle the problem, by stripping the control plane capability from the network devices and build a logically centralized control plane, called **controller**, so it will have a global view of the overall network. The controller is also abstracting the network infrastructure, so it gives a simpler logical view of the network, hence it is easier to program the network, even for complex traffic engineering.

The implementation of SDN, such as in OpenFlow-based SDN switch, also gives a flexibility in terms of the number of packet header fields that can be observed for packet classification. OpenFlow provides a capability to look at up to 38 packet headers fields [31] for each incoming flow.

ONF, a non-profit industry consortium, took a lead to standardize SDN. Figure 2.1 shows the SDN architecture stack defined by ONF:

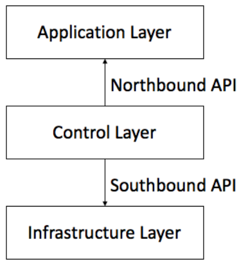


Figure 2.1: SDN Architecture Stack

The SDN stack is devided into 3 main layers: Application, control and infrastructure layer. Network devices reside in the **infrastructure layer**. The responsibility of infrastructure layer is to do traffic forwarding. Compared to traditional network, the intelligence of network devices is lower. The controlling responsibility resides in the **control layer**. This separation tally with SDN concept of decoupling the control plane from the data/forwarding plane. On top of simplified management, this separation also provides finer control over the traffic and more flexible forwarding decision based on the network conditions. **Application layer** consists of the operation tool and user interfaces to manage the network.

The communications between layers are done via the Application Programming Interfaces (API). There are two types of API, Southbound and Northbound API. **Southbound API** provides communication between Infrastructure Layer (Network Devices) and Control Layer (SDN Controller). **Northbound API** is used for communication between Application Layer (network tools and user interface) and the Control Layer.

Control Layer is the most important layer of the architecture. SDN controller, as the “brain” of the network, resides here. There are many SDN controllers available in the market now, from the vendor specific to open-source SDN controller, such as ONOS, NOX and OpenDaylight. OpenDaylight (ODL) [13] is used as base for developing many other controllers. It is a collaborative project by The Linux Foundation and supported by more than 20 big vendors.

2.2 Overlay Network

Overlay Network is a term for defining a virtual network that is deployed on top of an existing physical network (underlay network) . The overlay network allows us to have a constant view of the network, despite the change in the physical network, as shown on Figure 2.2. It consists of router, host and tunnels[2]. Tunnels are the overlaying mechanism of the paths in the underlay network, which provides the connectivity links in the overlay network. A single tunnel link is usually comprised of a set of links from the underlying network[2].

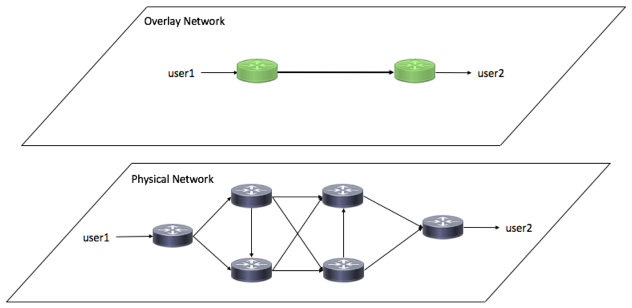


Figure 2.2: Overlay Network

There are some advantages on having overlay network, one of it is having a simpler logical network topology. Figure 2.2 shows that overlay network, with its virtual network, provides an abstraction of the physical network, hence the logical topology of the network becomes simpler. The simplification of logical network makes it easier to program the network. With a simplified network topology, the complexity to implement finer traffic engineering can be reduced as well.

In the SDN architecture, overlay network can fit very well due to its simplicity and capability to provides abstraction of the physical network.

2.3 Locator/ID Separation Protocol (LISP)

Locator/ID Separation Protocol (LISP) is a protocol in the network that separates network address into device identity, called Endpoint Identifier (EID), and device location, called Routing Locators (RLOC). LISP is different with current network addressing that combine both device identity and location into a single namespace. EID is the address of the enduser host, that is used for transport connectivity. RLOC is the routable IP addresses of the network attachment points that is used for routing in the transit network. Enduser host's EID can be reached after RLOC of the host is located. The main component of LISP is the mapping system, which is publicly accessible. Mapping system provides information of EID-to-RLOC mapping.

In LISP, unlike the RLOC, EID can be any address type, not limited to IP address. Other example of addresses that be used as EID are AS Number, Geo Location etc. This flexibility for the EID address is supported by the control message encoding in the LISP, named LISP Canonical Address Format (LCAF) [24]. It contains the information needed for communication between mapping system and other LISP components, that will be discussed later on.

The main problems that LISP is trying to solve with the separation is the routing scalability problem [17]. In the current system, with single namespace, the internet routing table grows rapidly. With the introduction of LISP, only RLOC is advertised to the internet. EID is assigned independently from the network topology and aggregated along the network boundaries. With this separation, since only RLOC is advertised to the internet, the internet routing table has lesser entries.

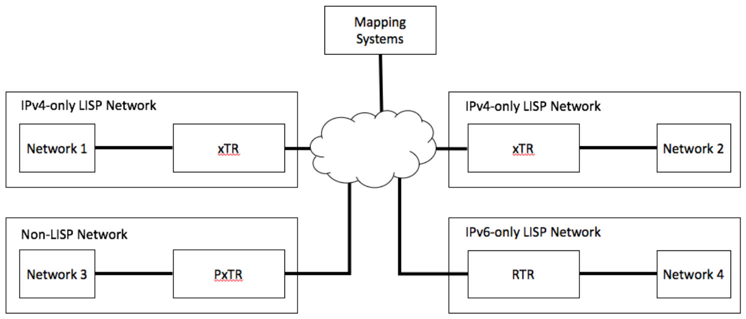


Figure 2.3: LISP

On top of that, separation EID and RLOC gives many other advantages, such as mobility, scalability, etc. By doing the separation, mobility can happen seamlessly by making the enduser host to be binded to a permanent address, EID. When enduser host location changes, LISP tunnel router will encapsulate the packets with a new RLOC and maintain availability the connection. By doing aggreation of the RLOC, scalability of the routing can be achieved as well. LISP also provides simplification for incremental transition towards IPv6, by allowing the deployment of IPv6 network on top of existing IPv4 infrastructure.

As mentioned above, The development of Open Overlay Router (OOR) is based on the LISP protocols and it's components. The terminology used by OOR is borrowed from LISP protocol, such as EID, RLOC, mapping system, etc. Below are the components of LISP protocol, as shown in Figure 2.3, the detailed functionality of each component will be discussed in the OOR section:

1. Mapping System: Mapping system is used to store the EID-to-RLOC mapping and as interface for lookup of EID location data when requested by the tunnel routers.

2. Ingress/Egress Tunnel Router (xTR): xTR is used for encapsulation of outgoing packet with LISP header (ITR) or decapsulation of incoming

packet's LISP header and send the packet to destination EID (ETR) in the communication between LISP networks. For example, in Figure 2.3, when a network 1 wants to send packets to network 2.

3. Proxy Ingress/Egress Tunnel Router (PxTR): PxTR acts like an xTR but for the non-LISP site that wants to communicate with LISP sites. For example, in Figure 2.3, when network 3 (non-LISP network) network wants to send packets with network 2 (LISP network)

4. Re-encapsulating Tunnel Router (RTR): RTR is used to decapsulation of the LISP header from the packets and re-encapsulate it again with different LISP header. For example, in Figure 2.3, when network 4 (IPv6-only) wants to send packets to network 1 (IPv4-only)

Chapter 3

Open Overlay Router

Overlay network is one of the known implementation of SDN paradigm, that provides the abstraction of the physical network. As discussed in the previous chapter, one of the advantage for overlay networks implementation is easier to program the network and reduce the complexity of performing finer traffic engineering. Open Overlay Router (OOR) [14] is an open-source platform to create programmable overlay network. It is written in C and support the implementation in Linux, Android and OpenWRT. Together with ODL LISPFlowMapping module[10], OOR supports the creation of overlay network and the configuration of it using NETCONF/YANG. Some features of OOR are[16]:

1. Multiple encapsulation in data-plane: VXLAN and LISP
2. Control-plane: NETCONF/YANG and LISP
3. Layer 3 overlay: IPv4 and IPv6
4. Multihoming, mobility, etc.

As mentioned above, the main functionality of OOR is to create programmable overlay network. The simple process of OOR is when a packet arrives, the overlay router will encapsulate it as overlay packet and route it via the underlay network to the destination. To be able to do that, the mapping system of the overlay and underlay infrastructure must be kept updated at all time.

In this section, we will discuss about the components of OOR and some use cases to show the life of packets in the OOR implementations.

3.1 Components

In this sub-section, more details about each component and their functionalities will be discussed. As mentioned in the previous section, the terminology used by OOR is borrowed from LISP protocol, including the naming for the components and the actions taken by those components. Not all of this components are required to implement Open Overlay Router, only Mapping System and xTR is mandatory. The rest of the components are optional, based on the need, condition and requirements of each deployment.

3.1.1 Mapping System

As mentioned above, mapping system is key aspect in OOR. There are two main components in the mapping system of OOR: Map-Server (MS) and Map-Resolver (MR). Mapping System is a service that can be accessed publicly and publish the mapping of EID-to-RLOC. Map-Server is the responsible for storing the mapping database and updating it according to the changes in the network. Map-Resolver is the interface for the lookup request (known as "map-request") sent by the xTR and responsible to do lookup of the EID location. When xTR is sending map-request, it will be routed to Map-Resolver that will forward it to Map-Server. This server can be anywhere, as long as it is accesible by the xTRs.

When there's a change in the overlay network, ETR/PETR will send Solicit Map-Request (SMR) [12] to the MS. Once MS received SMR, it will update their mapping database and let the ITR/PITR know that there's a change in the mapping, so ITR/PITR will update the cached mapping (called "map-cache", will be discussed in 3.1.2) in their system.

3.1.2 Ingress/Egress Tunnel Router (xTR)

Ingress/Egress Tunnel Router (xTR) works as edge router in the overlay network and used for communication between OOR networks. ITR will receive packets from the host, encapsulate it with LISP header and forward it to the ETR that corresponds to destination RLOC. ETR will decapsulate the LISP header and forward the packets to the destination host. During initialization, xTR will send their EID and RLOC information to the MS. When there's an incoming packet, ITR will send map-request to the Mapping System. Once ITR receives the mapping information, they will store it in the map-cache, encapsulate the packet and send it to destination ETR. Map-cache is subset of the mapping database of Mapping System in the xTR, the purpose is when another packets comes with destination EID that has been asked to Mapping System before, ITR can directly forward the packets to the destination without sending another map-request.

During the encapsulation process, LISP header is prepended on top of the original packet. When packet reachesLISP header consists of three parts[17]:

1. IP Header: The IP header carries the address information of the source and destination RLOC.
2. UDP Header: The UDP header carries the port number selected by the ITR during the encapsulation process. UDP is chosen because it provides enough information (port numbers) and it has zero checksum. There are already checksums in the IP header and, probably, TCP header, if the

original packet is using TCP protocol for the communication, hence the integrity of the original packet has been covered by that.

3. LISP-specific header: The LISP-specific header contains the flags and reachability information needed for LISP communication with fixed length (8 octets or 64 bits).

3.1.3 Proxy Ingress/Egress Tunnel Router (PxTR)

Proxy xTR is used to accomodate a situation where there are OOR and non-OOR sites. The way PxTR works is similar with xTR, the only difference is that PxTR is implemented as a gateway for non-OOR network to enters OOR network. When there are packets from non-OOR site coming to OOR site, PITR is responsible for encapsulating those packets with LISP header, sending map-request to Mapping System (if the EID-to-RLOC mapping is not available in the PITR map-cache) and forward the packets to destination EID, and vice versa for PETR. When one deploy a full OOR network, PxTR is unnecessary.

3.1.4 Re-encapsulating Tunnel Router (RTR)

In some deployment scenarios, there are two OOR networks but they can't communicate to each other directly. For example, OOR1 is using full IPv4 and OOR2 is using full IPv6. Re-encapsulating Tunnel Router is playing very important role to bridge these two networks. When packets from OOR1 travels to OOR2, RTR will re-encapsulate the IPv4 Packets from OOR1 with IPv6 header so the packets can be received by OOR2 ETR and forwarded to destination EID.

3.2 Sample Use Cases

In this sub-section, we will discuss about a use case of OOR to explain more about the packet life. The use case is for full IPv4-only OOR deployment. In the full OOR deployment, the required components are MS/MR and xTRs. Figure 3.1 shows the process happened when a packet is sent by user 1 (EID 1) to user 2 (EID 2), including the overview of the packet structure during the process. Below are the breakdown of the process happening in each steps:

1. When user 1 sends a packet, it will be directed to the xTR associated with user 1 (ITR), as it happened in the normal routing process.

2. xTR does not know the RLOC of EID 2, it sends map-request to the mapping system to ask for the mapping information for the EID of user 2.

3. Mapping system then forwards the map-request to the xTR associated with user 2 (ETR).

4. ETR sends the map-reply with the information of RLOC of the EID of user 2 to ITR. The map-reply is stored in ITR as map-cache, so when there's a similar request, ITR does not need to ask for the mapping again.

5. Once the ITR know where to forward the packet, it encapsulates the packet by prepending the LISP header on top of the original packet header and send it to the destination RLOC. The new IP header includes the RLOC of ITR as the source address and RLOC of the ETR as destination address.

6. Once the packet reaches the ETR, it will decapsulate the packet and remove the extra header added in the previous step and forwards the packet to user 2.

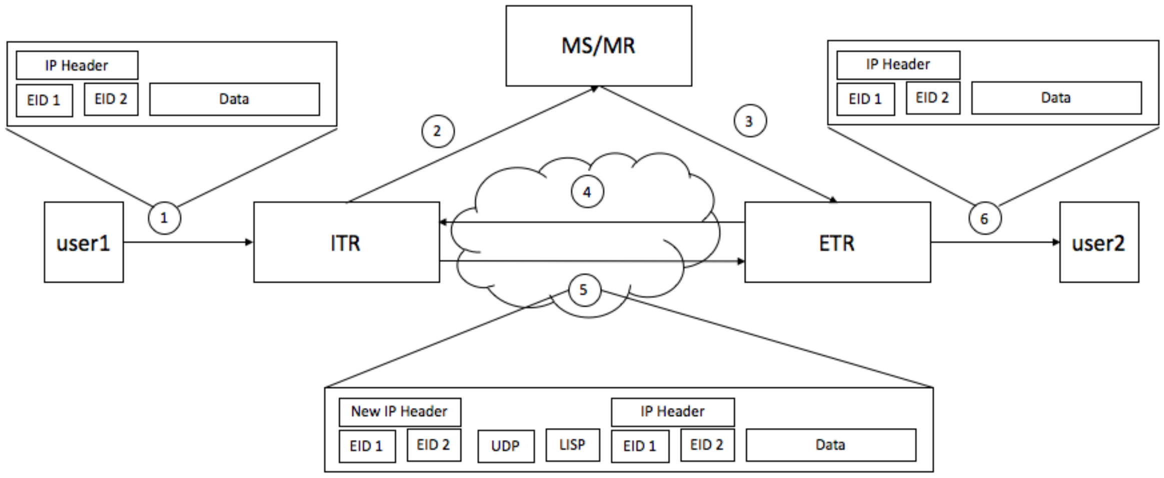


Figure 3.1: Full OOR Deployment Use Case

Chapter 4

Vector Packet Processing (VPP)

As discuss in previous chapter, OOR leverages LISP protocol for its control plane, such as for mapping update, retrieval etc. For the data plane, where the actual packet forwarding process happens, OOR is looking at Vector Packet Processing (VPP) platform to be integrated with the OOR platform. VPP is a high performance packet processing platform and runs entirely in the userspace (hardware and kernel agnostic). VPP was owned by Cisco Systems and is already running on many of the products in the market and recently the code was released to an open-source community called FD.io ("Fido") Project. Fido [21] is a collaborative open source project that focuses on implementing high-performance I/O for in the dynamic computing environments. It is a combination of several projects started with the Data Plane Development Kit (DPDK) to support programmability in the generic hardware.

VPP creates a superframe, vector of several packets available in the network. Once superframe is created, VPP processes the superframe through a directed nodes graph (Packet Processing Graph), as shown in Figure 4.1. This is different with most of the packet processing platform, that process the packet one by one in sequence through the whole graph. In VPP, the whole superframe is processed in a node, then forwarded to the next node. By doing so, the first packet in superframe is used to warm up the instruction cache in that node, so the next subsequent packets in the superframe can be processed in higher speed. With that approach, reliability is also increased. If there's a disruption or delay in the processing, the following superframe will have a bigger size, but the processing cost is still the same (only first packet in the superframe is used to warm up the instrution cache), hence the system can catch up to normal rate.

On top of the high performance and reliability, VPP is also very flexible and modular. The Packet Processing graph can be extended easily without changing the main code. For building an extension from a node (called as plugin), one can build it easily using a separate source code and load it to VPP's plugin

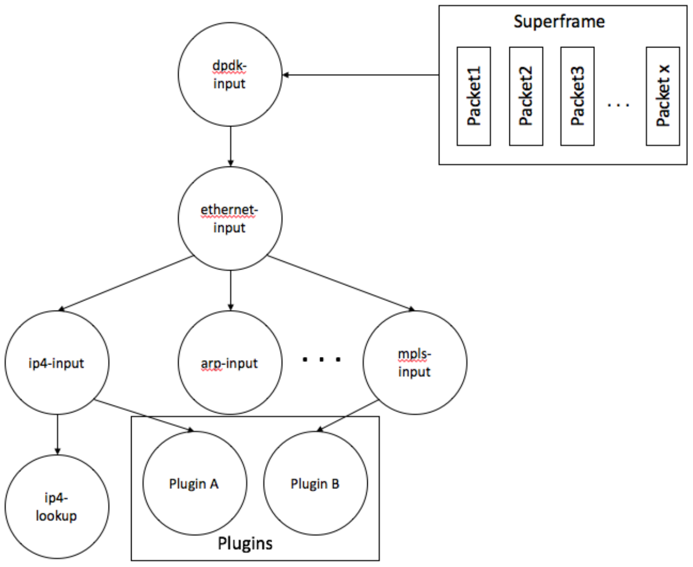


Figure 4.1: VPP Graph Node

directory. Plugin is introduced as a new node in the graph, as shown in Figure 3.1, hence there's no change in VPP main code.

4.1 Packet Classification in VPP

In VPP, packet classification is defined as a node, called "ip4-classify" (for IPv4) or "ip6-classify" (for IPv6) node. Packet classification is a process to find which flow that a packet belongs to and determine which action needs to be taken for the packet. VPP has a very fast packet classification model by introducing the mask-and-match model [23]. There are two main components in the VPP classifier, tables and sessions.

1. Tables: Tables contain a set of session with the same mask. Mask includes the specific byte positions to be checked by the system when the incoming packets arrive. The default action, in case of no rule matches the incoming packet, is also introduced here.

2. Sessions: Sessions are the the rules inserted by the users. It contains the packet header information, for example, source/destination address, port numbers, protocols etc, as the key and also the action that needs to be taken when the rule matches the incoming packet header. More than one sessions can be inserted in a table.

The following example can be used to better understand the process of packet classification in VPP. Say, a user wants to check the source address of all in-

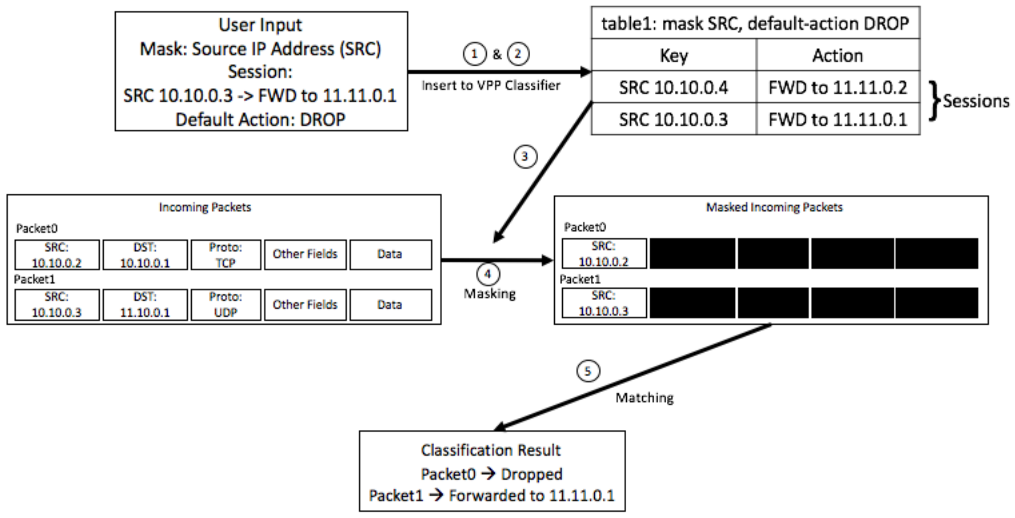


Figure 4.2: Example for VPP Classification Process

coming packets and if the source address is 10.10.0.3, the packet needs to be forwarded to router with IP address 11.11.0.1, else the packet will be dropped (default action). Below are the step-by-step process for the packet classification in that example scenario, also shown in Figure 4.2:

1. The user needs to create the table with mask to check only the source address in the IP header and define the default action. Let's say the table is called table1.

2. Once the table is created, session is inserted to that table. The session contains the source address of 10.10.0.3 and the action, forward to 11.11.0.1.

3. When packets arrive in the the "ip4-classify" node, the table informs the node to check the packet headers only for the defined in the mask, in this case only source address.

4. The node checks the source address of the packets. This process is called as masking. The rest of the fields in packet header are ignored.

5. After the masking process, the sessions in table1 are checked in sequence. If a match is found, the packet takes the action indicated in that session, otherwise, it takes the default action (drop in this case). This process is called matching.

The more details for the packet classification in VPP can be found in [17].

4.2 Why VPP?

As discussed above, VPP provides high-speed packet processing and also provides flexibility in terms of development for new plugin and management. The superiority of VPP compared to other well-known packet processing platform, such as Open vSwitch (OVS), has been tested by an independent test from European Advanced Networking Test Center AG (EANTAC) . In their report [22], EANTAC stated that VPP performance were "much more consistent and reliable than those of Open vSwitch". The detailed test result can be found in[22].

On top of that, VPP supports many APIs for management and programmability the platform. It supports low and high level API for the local and remote programmability that works via shared memory bus. VPP supports many high level API, such as netconf/yang, REST etc, so user can use any management agent/controller that their need. This is one of the features that will be beneficial in the future.

OOR choses VPP over OVS because of some other reason as well, such as, OVS is built to match and be compatible with OpenFlow. As mentioned above, OOR is using LISP, not OpenFlow, hence different approach needed. On top of that, OOR is still evolving, the flexibility of VPP is expected to help the evolution of OOR, such as with the fast packet forwarding and the flexibility of adding more plugins in the data-plane when needed, to support more advance feature development of OOR in the future.

Chapter 5

Development

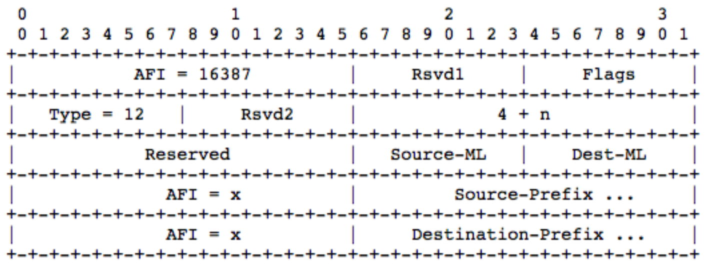
In the past, forwarding based on the destination address was enough for traffic engineering. The current Open Overlay Router (OOR) systems are based on destination address for packet forwarding and supports only IPv4 or IPv6 EID and RLOC. When a map-request is sent by the tunnel router, containing the destination EID, mapping system will do a lookup on its own mapping database and send back a map-reply to the tunnel router, with the destination RLOC IP address information for the tunnel router to forward the packet. On top of that, once map-reply is received by the tunnel router, it will be stored in the map-cache, so if the same destination information is needed, there's no need to ask the mapping systems again, hence, faster process and reducing the number of control message needed to forward a packet.

Due to the increasing number of application hosted in the network, more specific and flexible ruleset is needed. Because of that, the number of ruleset inserted to the systems is increasing. Based on the experimental setup, the current mapping and lookup systems can't scale really well to match the need of specific ruleset. In this thesis, a new 5-tuple LCAF is introduced based on the proposed multiple-lookup EID proposal [19], together with a proposal of new mapping systems. On top of that, to be able to scale more, rule aggregation and flow classification method are introduced and prototyped in the platforms.

5.1 5-Tuple LCAF Implementation

As mentioned in the beginning of this chapter, OOR only supports lookup and forwarding based on destination address, so only 2-tuple LCAF is supported. The datagram of 2-tuple LCAF is shown in Figure 5.1 below.

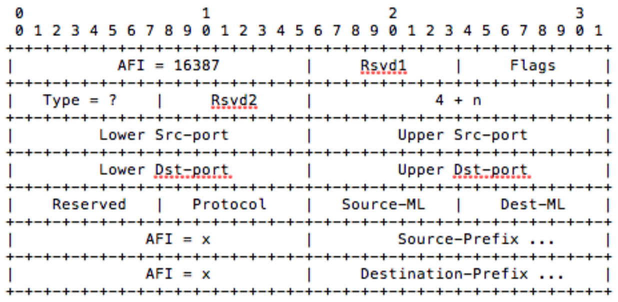
Due to the increased requirement for finer traffic engineering, 5-tuple lookup is needed, including the IP packet header information of source/destination address with their corresponding mask length, source/destination port numbers and protocol. To be able to achieve that, 5-Tuple LCAF is required to be implemented in OOR, following the proposed standard in [25]. The datagram



```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           AFI = 16387         |     Rsvd1     |     Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 12   |     Rsvd2     |             4 + n             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Reserved           |   Source-ML   |    Dest-ML    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              AFI = x          |         Source-Prefix ...     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              AFI = x          |     Destination-Prefix ...    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.1: Current LCAF in OOR (2-Tuple)

of 5-tuple LCAF is shown in Figure 5.2.



```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           AFI = 16387         |     Rsvd1     |     Flags     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = ?    |     Rsvd2     |             4 + n             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Lower Src-port        |         Upper Src-port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Lower Dst-port        |         Upper Dst-port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Reserved   |    Protocol   |   Source-ML   |    Dest-ML    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              AFI = x          |         Source-Prefix ...     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              AFI = x          |     Destination-Prefix ...    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5.2: Proposed 5-Tuple LCAF

To implement the the LCAF, a new structure is defined in the OOR for LCAF 5-tuple header, consisting of the information needed from the header. For now, the 5-tuple LCAF is registered as Type 17. The LCAF information is transmitted as part of the control message communication. By implementing the 5-Tuple LCAF, tunnel router can now send map-request with information of the addresses (including the mask length), source/destination port numbers and protocol. As seen on the Figure 5.2, there are lower and upper port numbers, in the current implementation, both of lower and upper port will have the same value. By adding the support for 5-tuple LCAF, it provides flexibility to implement application-specific or finer traffic engineering rules.

5.2 5-Tuple Mapping System

As mentioned above, the current mapping system of OOR is based on the mapping between destination EID and RLOC. To implement the 5-tuple or other lookup, a different structure of mapping database is needed. In this section, the proposed new mapping systems is discussed.

There are three types of databases implemented in the systems: Network

Information Base (NIB), Routing Information Base (RIB) and Forwarding Information Base (FIB). NIB stores the whole database of rules in the network. The rule is inserted directly to the OOR control plane or via SDN controller. NIB resides in the mapping systems of OOR. NIB also defines the default action, the action taken if the incoming packet cannot find any match rules in the NIB. RIB contains a subset of mapping from the NIB as a map-cache in the tunnel routers of OOR. The contents of RIB are achieved via the map-reply from mapping systems, when map-cache cannot be found. The last one is FIB, the mapping database in the data plane. FIB is the database used in the actual packet forwarding process. In the current OOR implementation, RIB and FIB is defined as single information base.

When there's an incoming packet, it will always first check the FIB for the mapping information. In the case of cache miss (map-cache cannot be found), it will send a map-request to the higher layer of information base, as shown in Figure 5.3. In the current OOR implementation, the 2-tuple lookup process is based on longest prefix match (LPM). Due to the complexity of implementing LPM for 5-tuple lookup, the proposed solution is using exact match as the lookup mechanism.

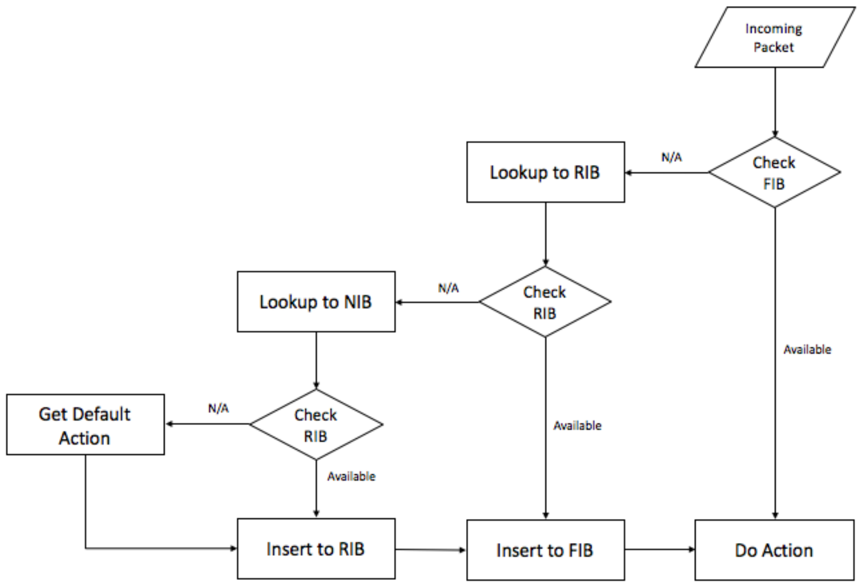
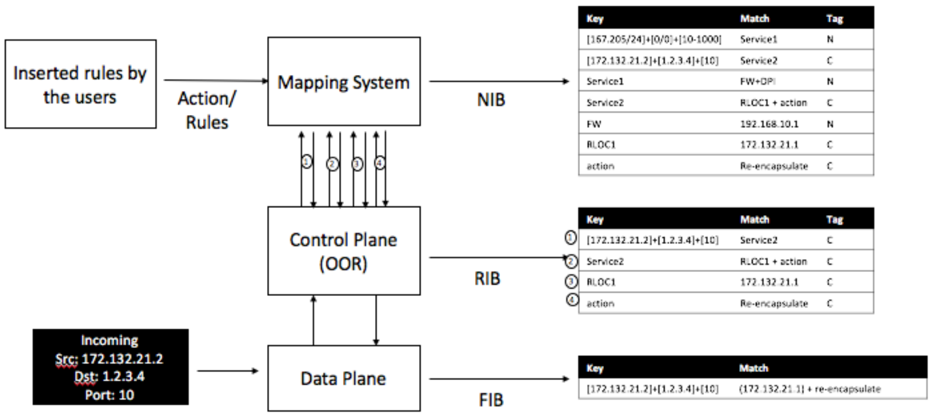


Figure 5.3: Mapping Lookup Process

The goals of proposed mapping systems are to provide faster lookup and forwarding process and to reduce the amount of map-request/map-reply message between databases. The database format includes the 5-tuple information as the key, match/action field and tag (for NIB and RIB). That format is used for the inserted traffic engineering rules, that consists of key and match/action information. The match/action field contains the destination RLOC and the action needs to be taken, such as forward or drop packet and re-encapsulate. The tag field is generated automatically to give information if the rules has been copied to the lower layer or not. The processes in the mapping system are broken down to three process: insertion, update and removal.

- **Insertion:** Traffic engineering rule is inserted in the NIB and added to RIB when there's a map-request. When there's a cache miss, RIB sends map-request to NIB and NIB sends all the rules that matches the source address. The reason behind that is source address is tied to EID behind the tunnel router, hence it is assumed that the rules with that source address will come up often. Afterwards, NIB updates the tag of all the copied rules as "copied to the lower layer" or "C", as shown in Figure 5.4.

- **Update/Removal:** In the event of removal or update in the NIB, the tag will be checked. If NIB sees that the tag of updated/removed rules as "copied to the lower layer", it sends Solicit Map-Request (SMR) to the RIB to inform that one or more rules has been updated/removed. If the rules are updated, RIB sends another map-request to ask for the updated rules, else it deletes the rules from its database. Otherwise, the removed or updated rules will only affect NIB.



Key	Match	Tag
[167.205/24]+[0/0]+[10-1000]	Service1	N
[172.132.21.2]+[1.2.3.4]+[10]	Service2	C
Service1	FW+DPI	N
Service2	RLOC1 + action	C
FW	192.168.10.1	N
RLOC1	172.132.21.1	C
action	Re-encapsulate	C

Key	Match	Tag
[172.132.21.2]+[1.2.3.4]+[10]	Service2	C
Service2	RLOC1 + action	C
RLOC1	172.132.21.1	C
action	Re-encapsulate	C

Key	Match
[172.132.21.2]+[1.2.3.4]+[10]	(172.132.21.1) + re-encapsulate

Figure 5.4: Proposed Mapping System

Regarding the update of FIB, it's same like above, except it will request for the rules from the RIB. Figure 5.4 shows the proposed mapping databases and the lookup process for an incoming packet. On the lookup from NIB to RIB, it will happen multiple times until the RIB get all of the information needed, as shown in Figure5.4, hence from a single request, RIB might get more than one new entry. When FIB requests a mapping from RIB, the control plane, where RIB resides, builds mapping data in the format that can be used directly by the FIB, that is later sent to the FIB for the actual forwarding process.

In this thesis, we are focusing on the mapping system with 5-tuple information. The lookup process for 5-tuple information is using exact match, hence the number of rules stored in the information bases can be very big. To tackle this concern, a new way to aggregate and classify flows are needed to reduce the number of stored rules and increase the performance in regards to the lookup process when the number of rules are big. In the next section, techniques for rule aggregation and flow classification are introduced.

5.3 Rules Aggregation and Flow Classification

With the increasing number of flows and rules needed, due to the increase of application hosted in the network, to be applied in the systems, the requirement for memory to store those rules are also increasing rapidly, together with the processing latency during the classification process. In this section, rules aggregation and flow classification method are proposed and discussed in details to tackle those problem. The aggregation algorithm is expected to be implemented in the NIB and classification is implemented in the FIB.

The implementation of aggregation and classification method is aligned with the current VPP classification method. In VPP, as mentioned in Chapter 4, the classification is done based on the mask-and-match model. Mask has to be pre-defined with specific length, hence matching for variable sized header is not supported. The basic concepts of our proposed solutions are splitting the rules into each packet header fields (5-tuple in our case, so 5 header fields), implementing multiple tables in the classifier system with multiple masks for each packet header fields, as a work around to accomodate wildcarding, and limiting the wildcard positions in the rule aggregation, such that it is positioned in sequence from the last bit, so the number of tables can be minimized.

5.3.1 Rules Aggregation

VPP does not support variable-sized header (wildcarding) mask to define the classifier table because the mask has to be pre-defined before creating the classifier table. Having wildcards in the rule means that they need a different mask with the rule with no wildcards. When users inserted a rule with wildcarding (netmask or range), rule needs to be broken down to some entries to remove the wildcards. For example, a rule with port number ranging from 16 to 22 needs to be broken down to 7 entries. When the number of rules increase, especially for the rules with wildcards, the requirement for space to store those rules also increase significantly, hence the system cannot scale.

The purpose of the proposed rules aggregation algorithm is to give users flexibility to use wildcards and minimize the number of entries that needs to be stored while still maintaining the accuracy of the rules. In this algorithm, the rules are splitted into each packet header fields, and then the wildcards in the aggregated entries are defined in sequence from the last bit of each header fields, so it can be predicted easily by the system. By defining the wildcard position in that manner, the possibility of the mask that needs to be defined to accomodate the rules can be minimized compared to if the position of wildcards can be anywhere. For example, as shown in Table 5.1, when there's a rule with the size of 4 bits, with this algorithm we only need to define 4 tables, instead of 15 tables if we let the wildcards to appear in any positions. Since the possible mask combination can be reduced, the number of tables can be minimized as well. Once the position of the wildcards are known, each entry is automatically inserted to a table with the suitable mask by the system. In the proposed algorithm, each rule is processed in the algorithm independently from the other rules.

Predicted Wildcards	Free Wildcards Position			
1111	1111	0111	1010	0010
1110	1110	1100	0101	0100
1100	1101	1001	0110	1000
1000	1011	0011	0001	

*1 means that the bit position needs to be checked in the classification process and 0 means that the bit positions can be neglected in the process
**0000 is not a valid mask since that means no bit positions need to be checked

Table 5.1: Number of Mask Possibility: Predicted vs Free Wildcards

To understand the algorithm, first we have to discuss about the variables in the algorithm. Rule R_i is the vector of entries E_j^i in binary format, where $i = \{1, 2, \cdots\}$ is the rule number and $j = \{1, 2, \cdots\}$ is the entry number. Each E_j^i has k members of sub-entries $e_{k,j}^i$ with k is each field information (for 5-tuple, $k = \{1, 2, \cdots, 5\}$) in the observed raw packet format. In the end, matrix R_i has size of $j \times k$.

$$R_i = \begin{bmatrix} E_1^i \\ E_2^i \\ \vdots \\ E_j^i \end{bmatrix} = \begin{bmatrix} e_{1,1}^i & e_{2,1}^i & e_{3,1}^i & e_{4,1}^i & \cdots & e_{k,1}^i \\ e_{1,2}^i & e_{2,2}^i & e_{3,2}^i & e_{4,2}^i & \cdots & e_{k,2}^i \\ \vdots & \vdots & \vdots & \vdots & \ldots & \vdots \\ e_{1,j}^i & e_{2,j}^i & e_{3,j}^i & e_{4,j}^i & \cdots & e_{k,j}^i \end{bmatrix}$$

When users insert rules to the system, the system automatically converts the rules to binary format. The length or size of the rules in binary format has been defined in [32]. Once we populated the matrix, we can start the aggregation process. R_i is inserted to Algorithm 5.1 and the result is $R_i^{'}$. In the algorithm, we leverage the fact that the entries are always ordered. The algorithm checks every entries from the first entry ($j = 1$) to the last and every packet header fields in each entry, from the first packet header field $k = 1$ until the last packet header fields in the entry ($k = 5$ in 5-tuple scenario). The bits in each $e_{k,j}^i$ are numbered and then processed from the last bit to the first. Variable $a = \{1, 2, \cdots, \lfloor \log_2 j - 1 \rfloor\}$ is introduced as a way to predict how many bits (from the last bit position) are possible to be aggregated, for example, if we have 5 rules, only the last 2 bits from each header fields are possible to be aggregated. By doing so, we can limit the computation only to the bit positions that are possible to be aggregated, hence the process can be faster. Once we have the *len* value, We also introduce the spacing variable s to couple the same bit position in two different entries, for example, for bit position 1, entry $j = 1$ is coupled with $j = 2$, for bit position 2, entry $j = 1$ is coupled with $j = 4$. The coupled bit position is compared, if they have two different value (0 and 1), then the bits in that bit position from j to $j + s$ are replaced with wildcards " $*$ ", otherwise do nothing.

Algorithm 5.1 Rule Aggregation

```
Input: $R_i$
Output: $R_i^{'}$
  $R_i^{'} = R_i$
  for $j = 1$ to $j$ do
    for $k = 1$ to $k$ do
      for $a = 1$ to $a = \lfloor \log_2 j \rfloor$ do
        $s(a) = 2^a - 1$
        $len = Length[e^i_{k,j}]$
        if $e^i_{k,j+s(a)}[[len - a - 1]] - e^i_{k,j}[[len - a - 1]] = 1$ then
          $R_i^{'}[j \ldots j + s(a)] = " * "$
        end if
      end for
    end for
  end for
  Remove all duplicated entries from $R_i^{'}$
```

For example, a user inserts a rule with parameters as shown in Table 5.2 (including the conversion to binary format and the size in bits):

Fields	Value	Binary	Size
Source IP Address	172.205.10.1	{1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,1 ,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,1}	32 bits
Destination IP Address	245.206.13.2	{1,1,1,1,0,1,0,1,1,1,0,0,1,1,1,0 ,0,0,0,0,1,1,0,1,0,0,0,0,0,0,1,0}	32 bits
Source Port Number	80	{0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0}	16 bits
Destination Port Number	20-24	{0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0} {0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0} {0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0} {0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0} {0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0}	16 bits
Protocol	TCP (6 [26])	{0,0,0,0,0,1,1,0}	8 bits

Table 5.2: Inserted rules by the users

The matrix R_1 for the sample rule above is (for simplicity reason, only last

4 bits of each field are written):

$$R_1 = \begin{bmatrix} \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{0,1,0,0\} & \{0,1,1,0\} \\ \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{0,1,0,1\} & \{0,1,1,0\} \\ \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{0,1,1,0\} & \{0,1,1,0\} \\ \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{0,1,1,1\} & \{0,1,1,0\} \\ \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{1,0,0,0\} & \{0,1,1,0\} \end{bmatrix}$$

Since the matrix is already populated, we can start the aggregation process. As seen from the example above, from 1 rule, we have 5 entries ($j = 5$), with different destination port numbers. The matrix R_1 is inserted into Algorithm 5.1. For this example, we are focusing on the destination port number only, since other fields do not need to be aggregated. Since we have 5 rules, only the 2 bits from each header fields are possible to be aggregated ($a = \{1, 2\}$), in this case bit number 4 ($a = 1$,$s = 1$) and bit number 3 ($a = 2$,$s = 3$). For bit position 4, entry j is coupled with $j+1$ and for bit position 3, entry j is coupled with $j + 3$. Figure 5.5 shows the coupling and wildcarding process.

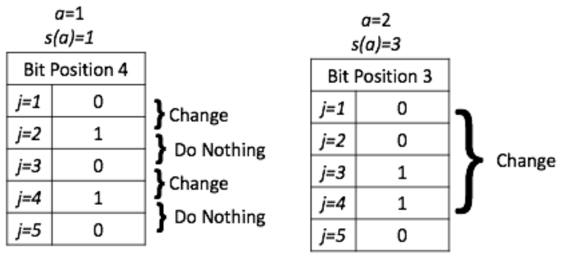


Figure 5.5: Coupling and Wildcarding Process

The result of the aggregation is shown as $R_1^{'}$ below:

$$R_1^{'} = \begin{bmatrix} \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{0,1,*,*\} & \{0,1,1,0\} \\ \{0,0,0,1\} & \{0,0,1,0\} & \{0,0,0,0\} & \{1,0,0,0\} & \{0,1,1,0\} \end{bmatrix}$$

$R_1^{'}$ can be aggregated to only 2 entries, hence the number of rules that is stored in the information base can be minimized without compromising the accuracy of the original rule.

5.3.2 Flow Classification

Flow classification is a process to find which flow that a packet belongs to and determine which action needs to be taken for the packet. The goal for implementing flow classification is to decrease processing latency as it will reduce the time needed to do lookup in the information base. The basic idea of the classification algorithm came from [20] and modified to better fit the requirement and the implementation in VPP. The main idea of this algorithm is to split the rules into smaller multiple lookup tables and to pinpoint the incoming packets to the specific lookup table with smaller set of rules that are already carefully selected, hence the lookup process can be more effective and the lookup time can be reduced.

As shown in Figure 5.6, the main process is the pre-filtering process. In pre-filtering, rules or incoming packets headers are checked in one or more pre-determined bit location to determine which lookup table to check for the mapping process. The process to determine the bit locations is called as effective bits selection. The bit locations are selected carefully by the system to make sure that it can split the rulesets into smaller multiple lookup table as even as possible. The position of bits selected is called Effective Bit Position (EBP).

The classification system consists of offline and online classification stages[20], as shown in Figure 5.6.

- Offline: When users insert a rule to the system, the system does pre-filtering for the inserted rules by the users, then insert the rule to most suitable lookup tables.
- Online: When packets comes to the system, the incoming packets is pre-filtered, then find the exact match from the candidate entries.

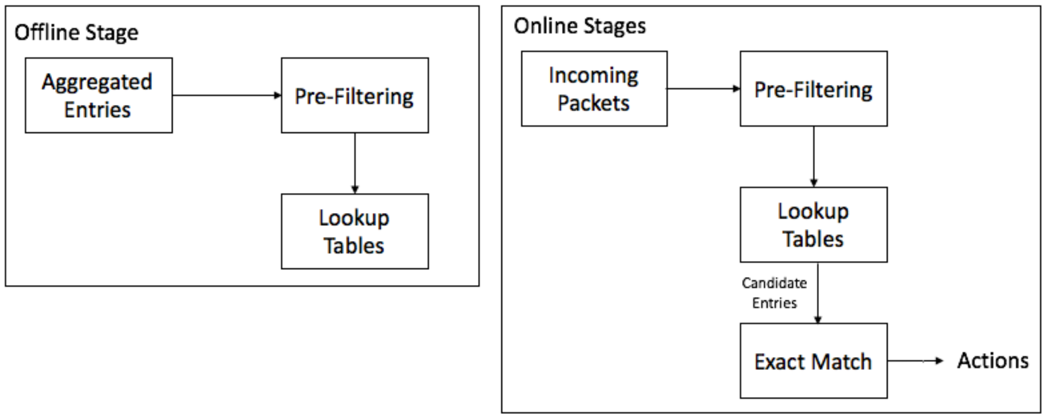


Figure 5.6: Proposed Classification Process

Below are the process to determine the effective bit position (EBP) used in the pre-filtering process:

- Find **Diversity Index** $H(q)$ for each bit position in each field: Diversity index is determined by calculating the entropy of each bit position of the entries stored in the information base, except the bit position that contains wildcard (“*”). The bit positions with high diversity index (high entropy) shows that the bits in that position has most even distribution of zeros and ones. That means if we filter the entries in the information base based on that bit position, the entries are distributed more evenly. By distributing the entries evenly, the number of entries in each lookup tables can be minimized. The formula for the calculation is shown in Equation 5.3.2, where q is the bit position, j is the total number of entries, and N_a^q is the total occurrence of bit a in a specific bit position q. Equation 5.3.1 shows the equation to calculate the probability p of each bit value showing in a specific bit position of q. If $p_1(q)$ or $p_0(q)$ is equal to 0, then $H(q) = 0$

$$p_a(q) = \frac{N_a^q}{j} \tag{5.3.1}$$

$$H(q) = \sum_{a=\{0,1\}} p_a(q) \log \frac{1}{p_a(q)} \tag{5.3.2}$$

- Calculate **Independence Index** Ind between bits position in all field: Independence index notates the correlation between bits positions, to ensure that if more than one bit chosen as the EBP, they have high independence. If two or more bits has high independence, it draws a good distinction between entries in the ruleset. Independence index is calculated as the mutual information between bits position, based on the diversity index calculation from the previous step. More number of effective bit means more number of lookup tables, hence lesser number of entries in each table. Equation 5.3.4 shows the calculation of Independence Index for the case where two EBP is needed, but it can be expanded to more EBP when necessary. To make the process faster, calculate Ind only for the bit positions with $H(q) > 0$.

$$H(q_\beta \mid q_\alpha) = \sum_{a=\{0,1\}} p_a(q_\alpha, q_\beta) \log p_a(q_\beta | q_\alpha) \tag{5.3.3}$$

$$Ind(q_\alpha, q_\beta) = H(q_\beta) - H(q_\beta \mid q_\alpha) = \sum_{a=\{0,1\}} p(q_\alpha, q_\beta) \log \frac{p(q_\alpha, q_\beta)}{p(q_\alpha)p(q_\beta)} \tag{5.3.4}$$

From both calculation, as mentioned above, the best result is when the bits position has high diversity index and, in the case of more than one EBP is required, high independence index between those bits position. Since the calculation results of Diversity Index and Independence Index can be conflicting, the priority is to select the bit position with highest diversity index first and another bit position that has high independence index with it. After the EBP calculation, when users insert rules, it will be pre-filtered and assigned to the most suitable lookup tables (offline stage) and when there's an incoming packet, it will also be pre-filtered, forwarded to the suitable lookup table find the exact match from the candidate entries inside that particular lookup table. If no match is found, then packet will take the default action.

In the beginning of implementation for new flow classification platform, we don't usually have any entries inside the information base. For this case, the EBP is chosen at random to get the system going. On top of that, EBP needs to be updated periodically, for example once a day, to ensure the validity of the EBP, especially after the addition or removal of many entries in the system. The goal of updating EBP regularly is to make sure that the number of entries in the lookup tables can be minimized most of the time.

5.3.3 Sample Use Case

Say that user is entering 3 rules (R_0 to R_2) in the system and Table 5.3 shows its aggregated entries $R^{'}$. Those aggregated entries is used to determine the Effective Bit Position. In this thesis, the number of effective bits chosen for the system is pre-determined manually. For this sample use case, 2 EBP will be selected, hence in total there will be 4 lookup tables. For the calculation of Diversity Index and Independence Index in this sample use case, Table 5.4 shows the value of variables used for the calculations. Table 5.5 shows the calculation for Diversity Index. From the calculation, we can see that $q = 3$ in source address and $q = 4$ in destination address has the highest Diversity Index, hence they are good candidates for EBP. Table 5.6 shows the calculation for Independence Index. In this calculation, we includes all q with $H(q) > 0$, in this case, only $q = \{3, 4, 5\}$ from source address and $q = \{3, 4\}$ from destination address. Based on the Independence Index, the pairing of $q = 3$ in source address and $q = 4$ in destination address has the highest Independence Index. From these calculations, we can decide that the EBP are to be the bit position 3 in source address and bit position 4 in destination port field.

No.	SRC Address	DST Address	SRC Port	DST Port	Protocol	Action
E_1	00011	0001$*$	$*****$	101$**$	00110	$action1$
E_2	00011	00100	$*****$	101$**$	00110	$action1$
E_3	00101	00010	$*****$	10110	10001	$action2$
E_4	00110	00010	$*****$	10110	10001	$action2$
E_5	00111	0000$*$	$*****$	101$**$	$*****$	$action3$

Table 5.3: Aggregated Entries of R_0 to R_2

Variables	Value
a	$\{0, 1\}$
q	$\{1, 2, 3, 4, 5\}$
j	5

Table 5.4: Variables for Calculations

In the offline stage, 4 lookup tables are created and the EBP for each entries is checked (pre-filtering). For example, for e_1, the EBP is 01, hence it will be inserted to lookup table 01. After all entries has been checked, the lookup tables will look as shown in Figure 5.7.

SRC Address	$a = 0$	$a = 1$	$H(q)$
$q = 1$	$p_0(1) = \frac{5}{5} = 1$	$p_1(1) = \frac{0}{5} = 0$	0**
$q = 2$	1	0	0**
$q = 3$	$p_0(3) = \frac{2}{5}$	$p_1(3) = \frac{3}{5}$	$\frac{2}{5} \log \frac{5}{2} + \frac{3}{5} \log \frac{5}{3} = 0.292$
$q = 4$	$\frac{1}{5}$	$\frac{4}{5}$	0.217
$q = 5$	$\frac{1}{5}$	$\frac{4}{5}$	0.217

DST Address	$a = 0$	$a = 1$	$H(q)$
$q = 1$	1	0	0**
$q = 2$	1	0	0**
$q = 3$	$\frac{4}{5}$	$\frac{1}{5}$	0.217
$q = 4$	$\frac{2}{5}$	$\frac{3}{5}$	0.292
$q = 5$	N/A*	N/A*	N/A*

SRC Port	$a = 0$	$a = 1$	$H(q)$
$q = 1$	N/A*	N/A*	N/A*
$q = 2$	N/A*	N/A*	N/A*
$q = 3$	N/A*	N/A*	N/A*
$q = 4$	N/A*	N/A*	N/A*
$q = 5$	N/A*	N/A*	N/A*

DST Port	$a = 0$	$a = 1$	$H(q)$
$q = 1$	0	1	0**
$q = 2$	1	0	0**
$q = 3$	0	1	0**
$q = 4$	N/A*	N/A*	N/A*
$q = 5$	N/A*	N/A*	N/A*

Protocol	$a = 0$	$a = 1$	$H(q)$
$q = 1$	N/A*	N/A*	N/A*
$q = 2$	N/A*	N/A*	N/A*
$q = 3$	N/A*	N/A*	N/A*
$q = 4$	N/A*	N/A*	N/A*
$q = 5$	N/A*	N/A*	N/A*

* The bit position has one or more wildcards, hence the $H(q)$ does not need to be calculated

** Since one of $p_a(q)$ is equal to zero, $H(q)$ is treated as being equal to zero.

Table 5.5: Calculation of Diversity Index

After that, there is an incoming packet as shown in Figure 5.8, called as *packet*1. The EBP is checked by the system then forwarded to lookup table 10 and checked as an exact match with candidate entries inside table 10. Since there's only one candidate entry e_5 inside table 10 and it provides an exact match with *packet*1 the action is retrieved, hence *packet*1 will do *action*3.

5.3.4 Implementation in VPP

We introduced a new plugin in VPP, called as "class-new" as the implementation of the proposed solution. The plugin supports the proposed flow classification method combined with the mask-and-match lookup method provided by VPP.

For the implementation in VPP, 4 EBP is selected, so the plugin has 16 set of classification lookup tables and one EBP table for pre-filtering. Each set of tables has ten tables, including tables for source/destination address with multiple netmasks, port numbers and protocol, so in total the classification plugin has 160 classification lookup tables and one EBP table. The breakdown for the table numbers is shown in Table 5.7.

The EBP table (Table 0) has two parts: EBP and entry number for each EBP. Entry number or N is the decimal representation of the EBP, with value

SRC Addr	DST Addr	$H(q_\beta)$	$H(q_\beta\|q_\alpha)$	$Ind(q_\alpha, q_\beta)$
$q_\alpha = 3$	$q_\beta = 3$	0.217	$\sum_{a=\{0,1\}} p_a(q_\alpha, q_\beta) \log p_a(q_\beta\|q_\alpha) = 0.253$	$H(q_\beta) - H(q_\beta\|q_\alpha) = -0.036$
$q_\alpha = 4$	$q_\beta = 3$	0.217	0.176	0.041
$q_\alpha = 5$	$q_\beta = 3$	0.217	0.176	0.041
$q_\alpha = 3$	$q_\beta = 4$	0.292	0.120	0.172
$q_\alpha = 4$	$q_\beta = 4$	0.292	0.311	−0.019
$q_\alpha = 5$	$q_\beta = 4$	0.292	0.311	−0.019

Table 5.6: Calculation for Independence Index

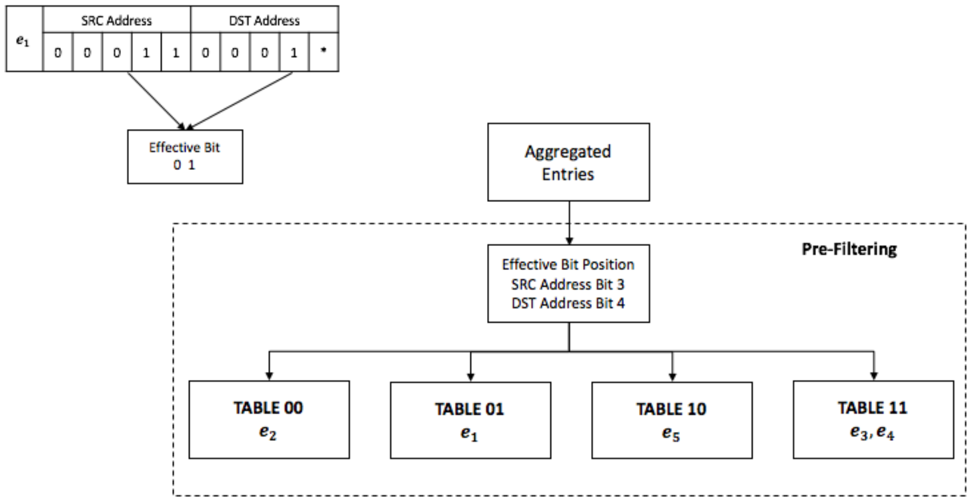


Figure 5.7: Offline Stage: EBP checking and lookup tables after pre-filtering

from 0 to 15. For example, if the rule or incoming packet has EBP 0000, the entry number N is 0 or if it has EBP 0001, N is 1 and so on. Entry number represents which set of classification lookup tables is the most suitable for that rule or packet. For example, if incoming packets has EBP 0101, means N is 5, based on Table 5.7, hence the set of tables of the be checked for the classification this this packet is from Table 56 to Table 65.

The decision for implementing ten lookup tables for each EBP was taken to tackle the flexibility problem in terms of wildcarding in the built-in classification. In the built-in classification module, if users wants to use wildcarding, such as netmask or ranges for IP address or port numbers, they have to create a new table with the masking that is suitable with their wildcarding requirement. Other alternative is break the rules into smaller entries manually and insert it one-by-one to the classification table. This limitation can increase the complexity for inserting a complex traffic engineering rules.

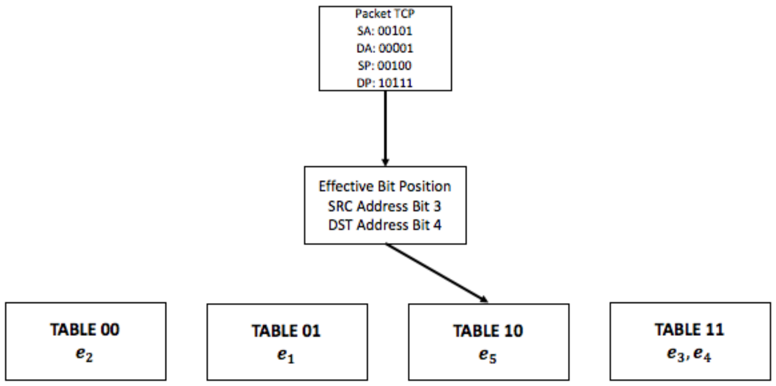


Figure 5.8: Online Stage

Table	Entries Stored
0	EBP
$1 + N * 11$ to $4 + N * 11$	Source addresses with different netmasks
$5 + N * 11$ to $8 + N * 11$	Destination addresses with different netmasks
$9 + N * 11$	Protocols
$10 + N * 11$	Source and Destination Port Numbers

Table 5.7: class-new table break down

In the new plugin, it creates all the multiple tables automatically with different mask for each header fields, for example, source and destination IP address has tables to accomodate /32, /24, /16 and /8 netmask. By doing so, for inserting rules with wildcards, the plugin decides which table that is most suitable for that rules with wildcards. If the plugin can't find the exact table that can accomodate the wildcarding of the rules, it will find the table with the closest mask and break it down to smaller entries so it can be inserted to that table, without compromising the accuracy of the original rules. For example, if the rule has the source IP prefix with /22 netmask, rather than creating a new table with mask to match that or break it down to many entries with /32 netmask, the system breaks the source IP prefix into 4 addresses with /24 netmask

The "class-new" plugin on VPP has some extra functions compared to the current VPP classification plugin, below are the most important one:

- class_gen: This function is used for the automation to create Table 0 with entries of the EBP, for pre-filtering, based on the entries from the previous section. When the new plugin is enabled, this function will generate the entries to represent the EBP. On top of that, this functions generates all the required lookup table-sets for the classification process.
- class_add_del_class: This function is enabled when users input the rules (Figure 5.9). It act as the offline stage of the classification process. When

there's new rule, this function automatically sends the rule to Table 0 for pre-filtering and get the entry number. Once it gets the entry number information, it inserts the rule into the lookup tables. The capability to choose the most suitable mask for rules with wildcards, as described above, is also included in this function.

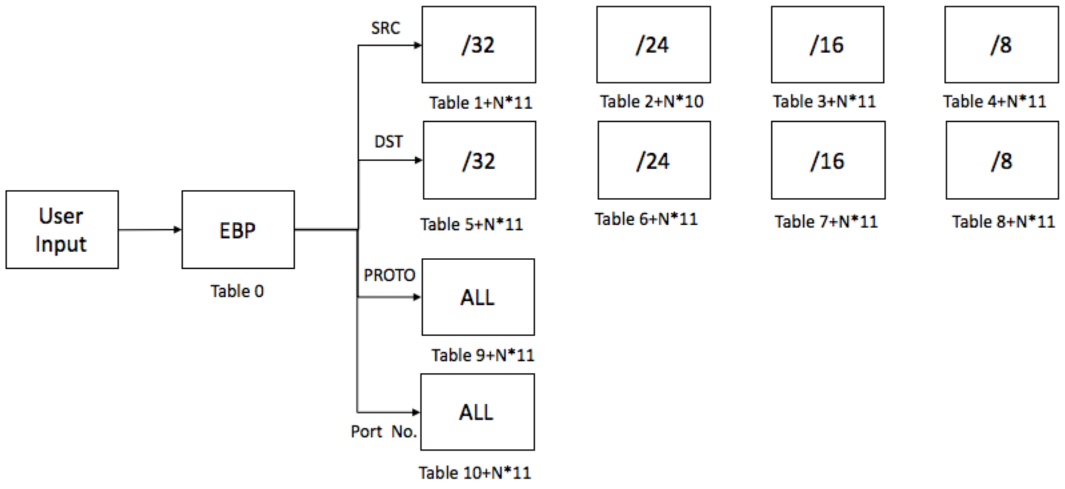


Figure 5.9: Classification offline stage at VPP

- class_node_fn: This function is used for the online stage of the classification process (Figure 5.10). When there's an incoming packet, this function checks if the packet has entry number information. If not, this function starts to do pre-filtering by sending the packet to Table 0 and re-circulates the packet to this plugin again with the entry number information. Otherwise it sends the packet to the most suitable lookup tables set (based on the entry number) and starts the matching process from in that tables (from source IP address until port numbers). If match is found, the packet takes the indicated action attached to the rule otherwise the packet is dropped right away (default action is dropping the packet).

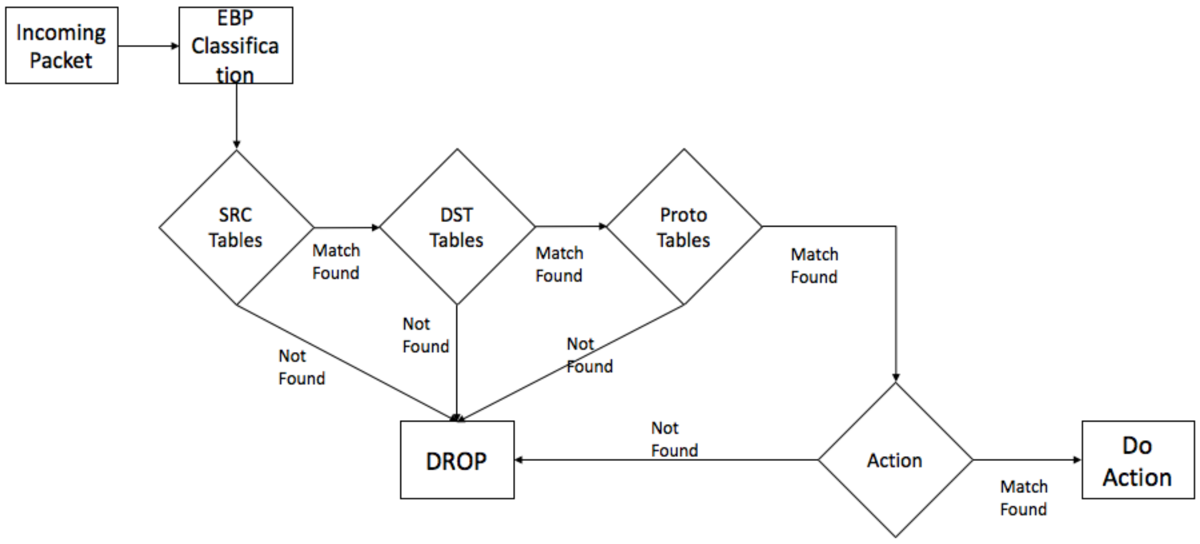


Figure 5.10: Classification online stage at VPP

Chapter 6

Results

In this chapter, we will discuss first about the simulation and testing setup together with the procedures and the tools used in the simulation and testing. Afterwards, the result of simulation and testing for the rules aggregation and flows classification method will be discussed.

6.1 Simulation and Testing Environment

The simulation were carried out with Mathematica to build the system-level simulation, to check if the proposed methods can works properly, before the implementation in the testbed. To deploy the testing environment in the testbed, four Ubuntu 14.04.3 virtual machines are used for different purposes:

- VM1: Used as border routers with five IP addresses to connect to the internet, management and connection to other VMs. VPP (version 16.09) and Classbench software resides in this virtual machines. The pre-filtering module is also implemented in this machine.
- VM2 and VM3: Used as the xTRs of OOR (version 1.0) setup and also for packet generator for the classification testing.
- VM4: Used as the mapping system of OOR and also packet generator for the testing.

All the results were taken from the simulation result (the simulation and implementation gave same result), except the lookup times results, it is taken from the implementation of the plugin in the testbed.

To generate the synthetic rules for the aggregation and classification, we used Classbench [10], a packet classification benchmarking tool. Classbench provides 12 real-world rules distribution, for the purpose of this thesis, 10 rules distribution are used. More information about Classbench can be found in [10].

For the packet generation, we used hping3 [11], a tool to generate ping-like custom TCP/IP packets. With hping3, we can generate TCP/IP packets with

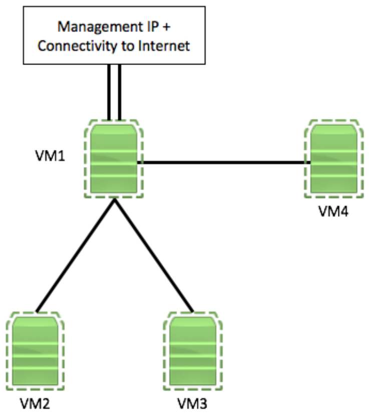


Figure 6.1: Testbed Setup

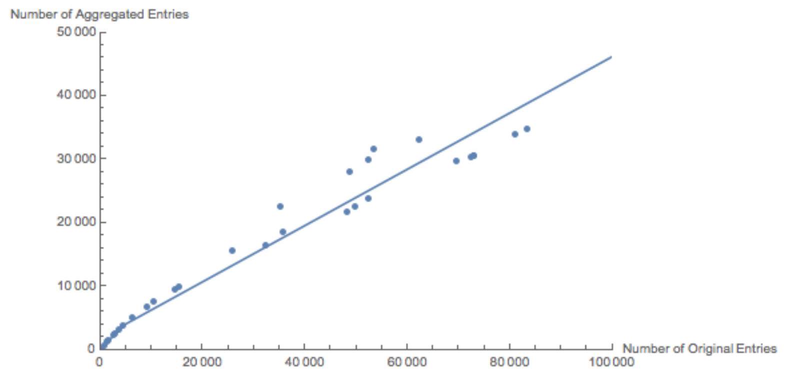
various headers information, including multiple protocols, port numbers, packet size etc.

6.2 Rules Aggregation

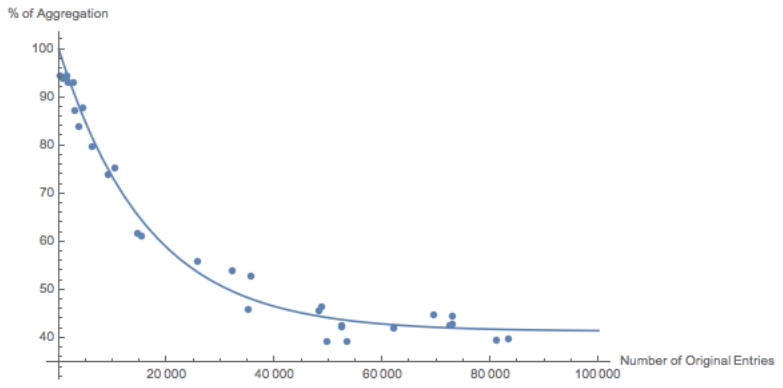
To observe the minimum achievable number of aggregated rules with the incorporation of the rule aggregation algorithm, we generated rulesets using 10 different rule distribution using Classbench, three sets for each rule distributions. The distributions that we used was ACL{1,3,4,5}, FW{1,2,3,5} and IPC{1,2}. The synthetic rules generated was processed with the rule aggregation algorithm before stored in the NIB.

From Figure 6.2, we can see that with incorporating the aggregation algorithm in the testbed, the number of entries stored in the NIB can be reduced. For example, from Figure 6.2 (a) when the number of original entry is 100000, it can be aggregated to less than 50000 entries. From Figure 6.2 (b), we can see as well that when the original entries are increasing, the rule can be reduced even more, for example, when we have 20000 entries, the aggregated entries is only around 58% of the original entry and for 60000 entries, the aggregated rules is around 42% of the original entry. In Figure 6.2 (b), we can see that the percentage of remainder of aggregated entries (compared to the number without aggregation) is distributed exponentially, to find the minimum achievable aggregated rules, FindMinimum function in Mathematica is used and the minimum achievable aggregated rules for this algorithm is around 41.4% of the original rules.

We can conclude that with incorporating the aggregation algorithm in the synthetic rules that is , we can reduce the space required to store the entries in the NIB up to 41.4% compared to the one without rule aggregation. By reducing the required space, cost can be reduced as well



(a)



(b)

Figure 6.2: Aggregation Result: (a) Aggregated Entries (b) Percentage of Aggregated Rules

6.3 Flow Classification

In this section, comparison was made between the basic classification, that only has a single lookup table to store all the traffic engineering rules, and the proposed classification method, with multiple lookup tables and pre-filtering by EBP, as dicussed in the previous chapter. In the basic classification, the rules were checked orderly in the table, whereas for the proposed methods, incoming packets were forwarded to a specific lookup tables with entries that has been pre-filtered. The aggregated entries from the previous section are inserted to the tables in this nodes with pre-calculated EBP. As mentioned above, once the entries are inserted to the tables in VPP, when there's an incoming packet, it will check the table and do the checking for the entries inside the tables in sequence.

In Figure 6.3, we can see the comparison of the number of entries that needs to be checked for the worst case scenario: the match is found in the last entry of the table. With the simulation, we can see that after around 30 entries, the proposed classification always checked less number of entries. In average only 56.9% of the aggregated entries is checked. The result as such was expected, since with the new algorithm, the rulesets are distributed evenly to multiple lookup tables, whereas without it, the rules are stored in the single big table.

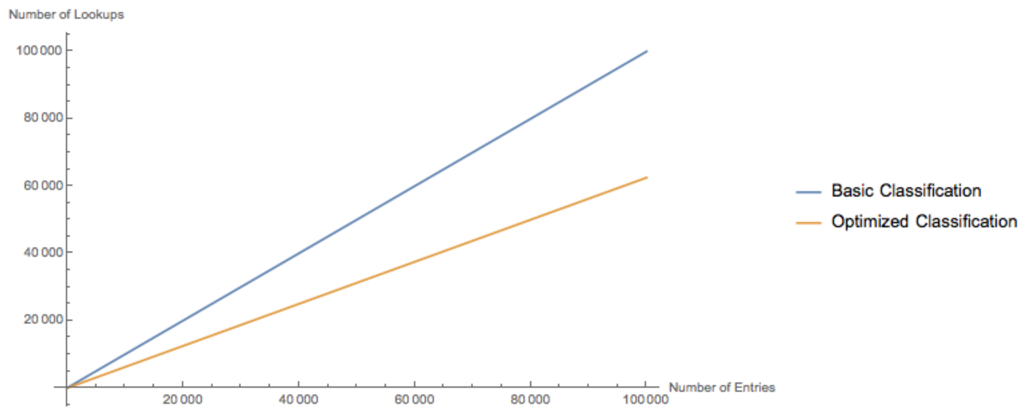


Figure 6.3: Number of Entries checked during the classification process

To prove the performance improvement with this algorithm, we did a test regarding the lookup times with real packets generated by hping3 in the testbed. The packets were generated simultaneously from VM2, VM3 and VM4 with random protocol and port numbers. Source and destination address was fixed since we have to make sure that the packets is coming from the VMs to VPP in VM1. The number of packets generated was 60 for each ruleset and the reading from the last 30 packets were taken. This step was taken to make sure that the system was already stable. Figure 6.4, shows that after 76 entries, by incorporating the aforementioned classification method, the lookup times is smaller compared to system without the classification method.

From the testing result above, we can conclude that by incorporating the classification method discussed in the previous chapter, we can minimize the packet processing time by the systems in regards to the lookup time, especially for big systems with large number of rules that needs to be inserted.

6.4 Analysis

From the result of simulation and testing shown above, we can see that by incorporating the rules aggregation algorithm in the mapping system helps to reduce the number of entries stored. The reduction of the stored entries correlates to lower the requirement for the storage. The flow classification in the data-plane helps to reduce the lookup times, hence lower processing latency. The combination of both exposes the ability to reduce the processing latency even more as

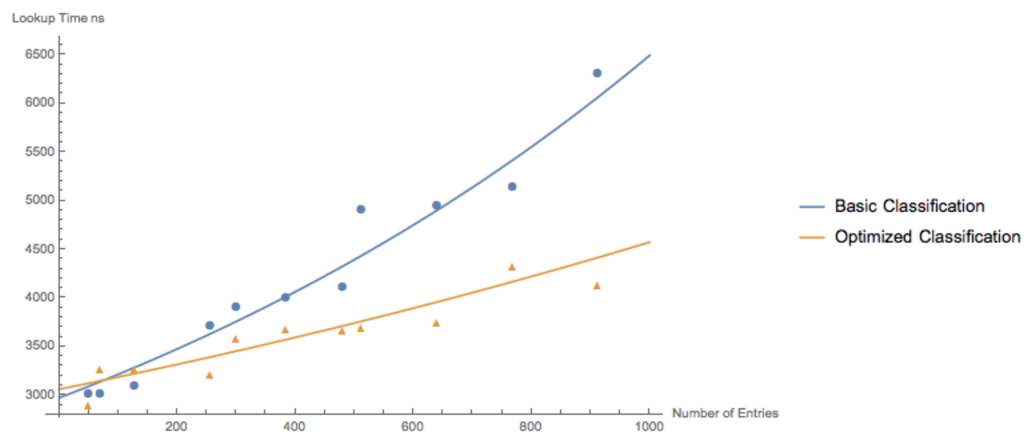


Figure 6.4: Lookup times comparison

well as further improving the scalability of the systems while still maintaining the precision of the routing system.

The rule aggregation helps to remove the redundant bits in the entries in systematic way, hence it can reduce the storage requirement, and still providing flexibility for multiple systems, even the systems that cannot support variable length header (wildcarding), such as VPP. Another benefit of the rule aggregation is that with smaller number of entries, the lookup times can be reduced as well.

By observing the aggregated entries, we can calculate the effective bit position (EBP) of the system. With EBP, we can split the entries into multiple smaller lookup tables evenly and forward the incoming packets into a specific lookup tables. As the result of that, the packets will only be checked against the pre-filtered rules, hence the decrease of the lookup times.

As shown in the previous section, implementation of both rules aggregation and flow classification methods with 5-tuple information results in up to 58.6% (maximum achieved) saving in the storage requirement and 29.6% (maximum achieved) reduction in the processing latency.

Chapter 7

Conclusions

7.1 Summary

The introduction of SDN paradigm gives a huge degree of flexibility in terms of the packet headers fields that can be observed for traffic engineering, for example as much as 38 packet header fields can be observed by OpenFlow-based SDN switch. SDN also decouples control plane from the network and create logically centralized control plane that gives the global view of the network, hence it is easier to implement finer traffic engineering. One of the well-known SDN implementation is overlay network, which provides network abstraction by building a virtual network on top of physical network, that simplify the logical topology of the network so the network is easier to program. Those capabilities come with costs, it does not scale well for huge number of rules, since the processing latency and requirement for storage to store the rules increases.

In this thesis, we investigated scalability problem faced by network owners because of the increasing number of applications hosted in the network - to provide granularity in terms of traffic engineering and quality of service with as low processing latency and storage requirement as possible, by incorporating the proposed rule aggregation and flow classification method. The proposed solution in this thesis was optimized for traffic engineering with 5-tuple information, since we believe that traffic engineering with 5-tuple information are adequate for majority number of network owners. The proposed rule aggregation was introduced to reduce the number of entries, hence it can minimize the storage requirement to store those rules. The proposed flow classification method was introduced to reduce the processing latency by splitting the rules into smaller multiple lookup tables and directing the incoming packets into specific lookup tables, that has been pre-filtered.

We brought the real life scenario into simulation and our testbed with multiple packet classification scenario. With the incorporation of the proposed rule aggregation algorithm, the number of entries was decreased, that corresponds to reduction of the storage space required by the system, compared to the system without the aggregation algorithm. The implementation of the proposed flow classification method in the system resulted in the reduction of the lookup time for the classification, compared to the systems with basic packet classification method. We also implemented the proposed solution for classification as a plugin for the Vector Packet Processing (VPP) platform.

The results indicate that the combination of rule aggregation and flow classification can decrease the storage space requirements as well as the processing latency for packet classifications, especially for systems with huge number of traffic engineering rules that needs to be implemented. A point to note is that the assessments performed here are based on simple assumptions and setups. Hence, the results should be seen as indication of the technology potential, rather than as definitive estimation of the real life performance.

7.2 Future Works

The works done in this thesis can hopefully lead to more researches in the packet classification area. We invite researchers to look more into the extension of the proposed aggregation and classification method.

The proposed rule aggregation is assuming that each rules are independent, no correlation with other rules, but there might be some duplicated entries because of rule overlapping. It is interesting to extend the proposed algorithm to tackle the rule overlapping problem so more space can be saved. On top of that, the algorithm is checking each bits of the rules and do simple arithmetic calculation, hence it is slow if the number of bits is high, such as in IPv6 implementation or when the number of field is increased. Therefore, it is important to find a faster way for the calculation inside the algorithm.

The extension of the flow classification method is also interesting to explore, such as another parameters to define the EBP and how to dynamically adjust the EBP and the lookup tables in real time. The calculation to determine the number of effective bit also interesting to explore, because in this thesis, the number of effective bit chosen as EBP is pre-determined, hence it might not be effective for large system. On top of that, the effectiveness of the proposed method with extension of more fields and in larger and more realistic setup and packets needs to be observed as well.

Bibliography

[1] J. Touch, "Dynamic Internet overlay deployment and management using the X-Bone", Computer Networks, vol. 36, no. 2-3, pp. 117-135, 2001.

[2] A. Parekh, "Routing on Overlay Networks", 2002.

[3] Open Networking Foundation (ONF), “Software-Defined Networking: The New Norm for Networks”, 2012

[4] V. Srinivasan, S. Suri and G. Varghese, "Packet classification using tuple space search", ACM SIGCOMM Computer Communication Review, vol. 29, no. 4, pp. 135-146, 1999.

[5] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015.

[6] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan and E. Porat, "On finding an optimal TCAM encoding scheme for packet classification", 2013 Proceedings IEEE INFOCOM, 2013.

[7] T. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching", ACM SIGCOMM Computer Communication Review, vol. 28, no. 4, pp. 203-214, 1998.

[8] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane and P. Eugster, "SAX-PAC (Scalable And eXpressive PAcket Classification)", ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, pp. 15-26, 2014.

[9] S. Singh, F. Baboescu, G. Varghese and J. Wang, "Packet classification using multidimensional cutting", Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM ’03, 2003.

[10] B. Lantz, B. Heller and N. McKeown, "A network in a laptop", Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets ’10, 2010.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow", ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, p. 69, 2008.

[12] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings", IEEE Micro, vol. 20, no. 1, pp. 34-41, 2000.

[13] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi and C. Yim, "Boundary Cutting for Packet Classification", IEEE/ACM Transactions on Networking, vol. 22, no. 2, pp. 443-456, 2014.

[14] A. Rodriguez-Natal, "Open Overlay Router Wiki", GitHub, 2016. [Online]. Available: https://github.com/OpenOverlayRouter/oor/wiki. [Accessed: 05- Aug- 2016].

[15] "OpenDaylight Lisp Flow Mapping:Architecture - OpenDaylight Project", Wiki.opendaylight.org, 2016. [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Lisp_Flow_Mapping:Architecture. [Accessed: 05- Aug- 2016].

[16] A. López Brescó, "Open Overlay Router Features", GitHub, 2016. [Online]. Available: https://github.com/OpenOverlayRouter/oor/wiki/Features. [Accessed: 05- Aug- 2016].

[17] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, January 2013.

[18] "The OpenDaylight Platform | OpenDaylight", Opendaylight.org, 2016. [Online]. Available: https://www.opendaylight.org/. [Accessed: 05- Aug- 2016].

[19] N. Chowdhury and R. Boutaba, "A survey of network virtualization", Computer Networks, vol. 54, no. 5, pp. 862-876, 2010.

[20] C. Hsieh and N. Weng, "Many-Field Packet Classification for Software-Defined Networking Switches", Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems - ANCS '16, 2016.

[21] "The Fast Data Project", Fd.io, 2016. [Online]. Available: https://fd.io. [Accessed: 05- Aug- 2016].

[22] Light Reading and European Advanced Networking Test Center AG, "Validating Cisco's NFV Infrastructure", 2016.

[23] "VPP/Introduction To N-tuple Classifiers - fd.io", Wiki.fd.io, 2016. [Online]. Available: https://wiki.fd.io/view/VPP/Introduction_To_N-tuple_Classifiers. [Accessed: 05- Aug- 2016].

[24] D. Farinacci, D. Meyer, and J. Snijders, "LISP Canonical Address Format (LCAF)", RFC Draft, July 2016.

[25] A. Rodriguez-Natal et al., "LISP support for Multi-Tuple EIDs", RFC Draft, January 2016

[26] "Protocol Numbers", Iana.org, 2016. [Online]. Available: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml. [Accessed: 05- Aug- 2016].

[27] "ClassBench: A Packet Classification Benchmark", Arl.wustl.edu, 2003. [Online]. Available: http://www.arl.wustl.edu/classbench/. [Accessed: 05- Aug- 2016].

[28] "hping3(8) - Linux man page", Linux.die.net, 2016. [Online]. Available: http://linux.die.net/man/8/hping3. [Accessed: 05- Aug- 2016].

[29] http://www.cs.huji.ac.il/~feit/papers/exp05.pdf

[30] A. Feldman and S. Muthukrishnan, "Tradeoffs for packet classification", Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), 2000.

[31] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[32] https://tools.ietf.org/html/rfc790

[33] B. Pfaff, "The design and implementation of Open vSwitch", Proc. USENIX Symp. NSDI, pp. 117-130, 2015

[34] B. Vamanan and T. Vijaykumar, "TreeCAM", Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies on - CoNEXT '11, 2011.

[35] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown and S. Shenker, "NOX", ACM SIGCOMM Computer Communication Review, vol. 38, no. 3, p. 105, 2008.

[36] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, “Fabric:A retrospective on evolving SDN”, Proceeding in 1st Workshop Hot Topics Software Defined Networking, pp. 85-90, 2012.

TRITA TRITA-EE 2016:148