Forwarding Table Entries in Software Defined Networks: Representation and Uses in Network Engineering

by

Liang Yang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Network Engineering.

Victoria University of Wellington
2018

Abstract

Software Defined Networking (SDN) is an emerging architecture that decouples the network control and forwarding functions. In SDN, the functionality of static configuration and routing table in a traditional network has been replaced by forwarding table entries (FTEs). Thus a systematic research on FTE to better monitor traffic and manage networking resources becomes crucial in SDN. There are already some initial works on FTE representation from mathematical/logical perspective. However, they usually concentrate on the abstraction and expression of FTE rather than the applications in real network. Based on existing research, a controller is unable to monitor networking traffic and manage networking resources from a network-wide perspective. To address these challenges, Boolean algebra is chosen and extended in this thesis to examine the relations and manipulations among FTEs together with traffic statistics. Specifically, three SDN applications: i) equivalence evaluation during FTE deployment, ii) non-invasive traffic estimation and iii) anomaly detection, have been proposed and verified with the help of Boolean algebra. All of these applications rely on the mining of the FTEs and their associated statistics, thus no overhead will be introduced to the switch's original packet forwarding functionalities. They can be easily deployed in production networks due to the non-invasive strategy as well as the feasibility and flexibility in real networking scenarios.

Acknowledgements

I would like to express my special appreciation and thanks to my primary supervisor Dr. Bryan Ng, you have been a tremendous mentor for me. Your advices on both research as well as on my career are priceless. I would also like to thank my secondary supervisor, professor Winston Seah, for your insightful comments and encouragement, but also for the hard question which motivated me to widen my research from various perspectives. I also learned a lot mathematical thinking and writing from my third supervisor, Dr. Lindsay Groves, who always gave me thoughtful comments and sometimes they are even longer than my paper.

A special thanks to my family. Words cannot express how grateful I am to my mother-in law, my mother and my father for all of the sacrifices that you've made. I would also like to thank my son who was always willing to accompany me in the lab to do research in weekends. At the end I would like express appreciation to my beloved wife, Ning Fei, who was always my support in the moments when I felt hopeless in my PhD journey.

Publications

- Liang Yang, Bryan Ng and Winston KG Seah, *'Heavy Hitter Detection and Identification in Software Defined Networking"*. Proceedings of 2016 25th International Conference on Computer Communication and Networks (ICCCN), 2016, Waikoloa, HI, USA.
- Liang Yang, Bryan Ng, Winston K.G. Seah and Lindsay Groves, *"Equivalent Forwarding Set Evaluation in Software Defined Networking"*. Proceedings of the 15th IFIP/IEEE International Symposium on Integrated Network Management (IM), 8-12 May 2017, Lisbon, Portugal.
- Liang Yang, Bryan Ng, Winston KG Seah and Lindsay Groves, *"Deterministic Confidence Interval Estimation of Networking Traffic in SDN"*, Proceedings of the 42nd IEEE Conference on Local Computer Networks (LCN), October 9-12, 2017, Singapore.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software-Defined Networking (SDN) offers a way to achieve the network agility required by increasingly dynamic environments such as networking virtualisation and cloud computing. SDN decouples the network control plane from the network forwarding hardware, and moves the control logic and state to a programmable software component, *the controller*. In essence, SDN is a logically centralised design which focuses on network-wide visibility and control rather than device-level configuration and management [1]. Compared with traditional networking, this architecture offers better administrative scalability which is the ability to accommodate the communication growth as a network expands.

A core concept of SDN is that by creating a few careful abstractions, the complexity of the underlying components is hidden and new solutions are easily developed via the programming interface between the network devices and the controller. One popular interface is OpenFlow, the de facto standard that gives access to the forwarding plane of a network [2]. In OpenFlow, forwarding table entry (FTE) has replaced routing table entry in traditional networking to manage the forwarding capabilities of switches and routers. Packets are forwarded according to the path which is composed of these entries. For a software defined network, the behaviours of all packets can be precisely controlled by manipulating FTE.

This is the motivation for the research on FTE.

1.1 Motivation

Traditional networks rely on routing protocols to define the forwarding path for packets while SDN manages the network via FTEs. In SDN, the networking forwarding functionalities are implemented by the manipulations of the FTEs. FTE is well organised but also complicated in terms of component structure and functionalities. Based on the latest OpenFlow specification [2], there are multiple flow tables inside a switch, and each FTE contains match fields, instructions, priority, etc. An incoming packet can be modified, forwarded or dropped according to the instructions of a matching FTE. Moreover, each FTE is associated with a statistic allowing the controller to monitor the network. From the view of controller, it has a global view of all the FTEs in all switches. Hence, there is a need to find a systematic method to abstract, analyze and manipulate the FTEs and further apply them into the network management.

An intuitive solution to devise an FTE representation is to find a suitable mathematical approach and then apply it to FTEs. Some researchers have already attempted to do this with logical connective [3, 4] or abstract algebra [5–14]. Unfortunately, these existing works are constrained by the selected mathematical or logical methods and fail to take into account major FTE attributes especially priority and multiple flow tables. Since it is unlikely to find an exact matching model to express all these key attributes, an adaptation and expansion of the existing methods will be required.

The goal of a representation is to better understand and utilise the functionality of FTEs inside a software defined network. An effective FTE representation must meet the following criteria: (i) it should cover the major attributes of FTE, for example, wildcard, priority and multi-table; (ii) this representation is capable of exploiting the various manipulations on

FTEs such as validation and facilitating FTE placement; and (iii) the representation should have the ability to integrate network information such as traffic statistics and topology to resolve realistic networking problems.

In a word, compared to traditional distributed networking, a controller in SDN has the ability to monitor a network's status and manage the resources. This is achieved by the understanding and manipulation of the FTEs. However, the correctness, robustness and reliability of FTE placement and optimasation must be guaranteed by a systematic method. This requirement inspires the first motivation: find a way to represent the core attributes of FTE. With the help of this representation, a controller can easily transfer high-level policies to low-level FTEs and correctly place them into switches or routers. Moreover, SDN does not only offer the convenience of configuration and management, it enables the possibility to dynamically regulate a network's resources. The typical applications include the statistics estimation, load balancing and anomaly detection. These objectives cannot be accomplished without a network's statistics and topology information. A good representation should be easily integrated with these information to solve realistic networking problems. Thus the utilisation of a representation in network engineering becomes this thesis's second motivation.

1.2 Research Scope

An FTE representation can be achieved by adopting either top-down (high-level) or bottom-up (low-level) strategy. A top-down approach starts with a big picture and breaks down from there into small segments. A representation based on this approach is suitable for representing the fundamental mechanism of FTE. Instead, a bottom-up approach builds up a system from the analysis of the detailed functionalities of each component and then tries to put all pieces together. The representation designed by a bottom-up approach is good for describing FTE's networking func-

tionalities. Take the decision tree model as an example, every single node represents an individual packet forwarding rule while the links between nodes express the relations among all the tables in a switch [15].

The research scope of this thesis is illustrated in Fig. 1.1 which indicates the scope mainly falls into low-level representation and multiple flow table applications (the second quadrant). Both the third and fourth quadrants focus on the applications of individual forwarding tables or switches with the help of either high-level or low-level representation. As indicated by the blank first quadrant, currently there is no research which fully represents FTE with a high-level representation while achieving network-wide applications.

The low-level representations focus on the specific FTE attributes while the high-level representations emphasise on the overall functionality of a network. The most widely used low-level FTE representations include rule decomposition/composition (DIFANE [16] and Palette [17]), graph theory (CacheFlow [15] and EAR [18]), decision tree model (FlowAdapter [15], CAB [19], vCRIB [20] and Maple [21]), rectangular/brick representation (One Big Switch [22] and RuleBricks [23]). These approaches are good at expressing the attributes of a single FTE as well as the relations among multiple FTEs inside one table, thus they are suitable for investigating the issues such as redundancy and routing loop in an individual table. They are also used to optimise the placement of FTE in a single table.

As opposed to the low-level representations which pay more attention to the structures and details of an FTE, the high-level representations attempt to interpret the FTEs from the perspective of a network. They use either formal mathematical logics, for example, Kleene algebra (NetKAT [5]), first-order logic (FLOVER [9]), propositional logic (NICE [14]), set theory (Frenetic [7] and PathQuery [24]), or high-level programming languages, for example, functional programming (Nettle [25]), declarative programming (Pyretic [26]), domain-specific language (Featherweight [13,27] and XML/NML [28,29]) to approximate or express the for-

warding behaviour of a software defined network.

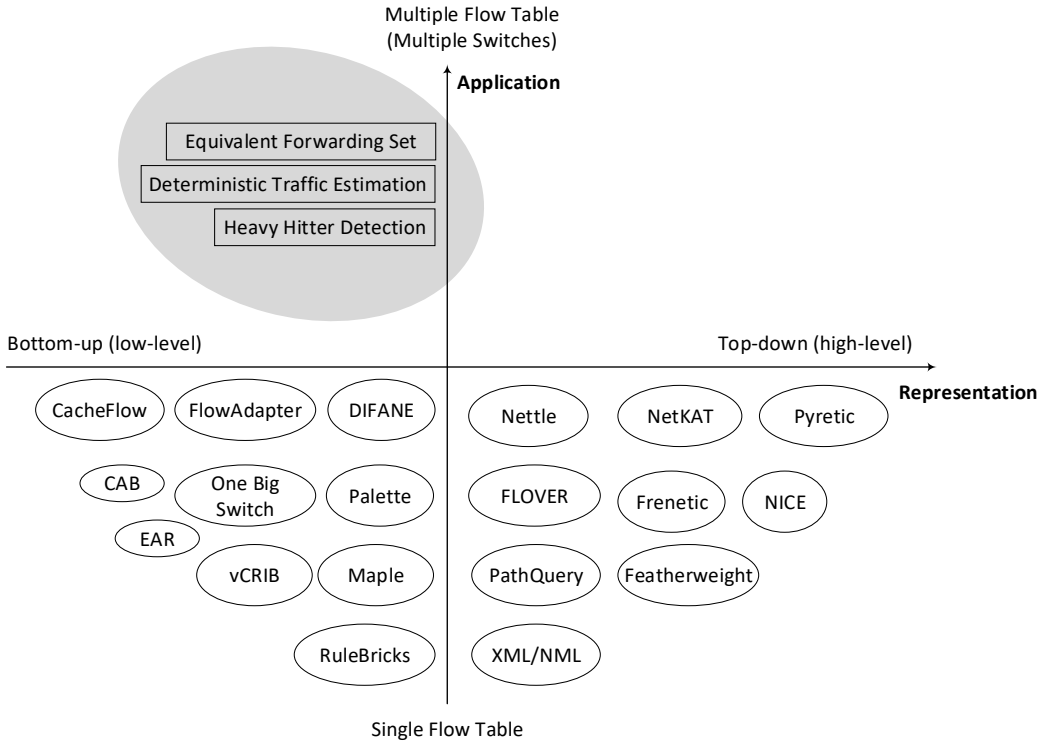


Figure 1.1: Research scope & contributions

Unlike the traditional networking which requires a network administrator to statically configure a network at device-level scale, SDN provides the ability to dynamically program networking devices at network-wide scale, which means better scalability can be achieved by SDN. Hence an ideal SDN representation is able to reflect the network-wide functionalities as well as to cover all the attributes of an FTE. To the best of my knowledge, there is no such existing representation method to achieve both objectives. This thesis does not intend to fill that gap by inventing a new formal method to fully represent all attributes of a software defined network, it endeavours to develop practical SDN applications based on the selected FTE representation.

1.3 Research Questions

Forwarding table entry is a bridge between a controller and a switch, it bears the responsibility for forwarding packets and reporting traffic statistics, a better understanding and manipulation of the forwarding table entries to explore more applications become critical. The core research questions of this thesis are: "***Is there an effective way to represent the forwarding table entries in SDN ?***" and "***How to exploit this representation in network engineering to offer better or novel solutions to manage and monitor large-scale networks.***". More specifically, the following research questions are expected to be answered in this thesis:

- How to evaluate the forwarding behavior of the various types of table(s)? Given two different types of tables, is there a systematic method to determine whether they achieve the same forwarding functionalities?
- How to estimate any given flow's statistics without impacting the existing forwarding functionalities? This method should not require any new FTE installation.
- How to detect heavy flow traffic without introducing any overhead on switch side? This approach should not require the switch to send any sampling packets or do any hardware customisation.

1.4 Research Contributions

The overall goal of this thesis is to explore the capability of SDN by finding and applying a suitable FTE representation. With the adaptation and expansion of Boolean algebra, the following three solutions are investigated and implemented.

- **Equivalent Forwarding Set Evaluation**
  In SDN, the forwarding pipelines are constructed either in the form

of a single table or multiple linked tables. By converting any given arbitrary forwarding table(s) into a uniform representation called *equivalent forwarding set*, the process of evaluating equivalence between two forwarding sets in terms of networking functionality are formalised and implemented. It facilitates the flow table management in the controller as well as the FTE placement in the switch.

- **Non-invasive Deterministic Statistics Estimation**
  For a flow with a matching FTE in a switch, its statistic is easily acquired by a status inquiry from a controller to the switch on this flow's corresponding FTE. But for the traffic estimation on the flow whose corresponding FTE does not exist, its statistic is not known until a new FTE is installed for the purpose of monitoring. However, the extra packet forwarding delay and the decrease of throughput as well as the potential conflicts between the monitoring FTEs and the existing FTEs jeopardise the feasibility of this approach. To avoid these drawbacks, a non-invasive traffic estimation approach based on the existing FTEs' statistics has been implemented and verified. This traffic estimation solution does not affect the existing packet forwarding functionality and performance, which makes it more practical and suitable for large-scale data centre networks.

- **Heavy Hitter Detection**
  In a large network, it is often important to be able to detect high-volume traffic (heavy hitter). A new heavy hitter detection approach which relies on mining traffic statistics (e.g. port bitrate) and FTE across multiple switches is designed and implemented. Comparing to the existing sampling-based and sketch-based detections, this approach simultaneously achieves considerable accuracy and good scalability.

1.5 Research Framework

All the aforementioned contributions rely on the same representation - Boolean algebra. Boolean algebra is naturally suitable for FTE because the match fields and their interactions are easily expressed in the form of Boolean function. Figure 1.2 presents the adopted research framework in which three applications are developed with the help of Boolean algebra.

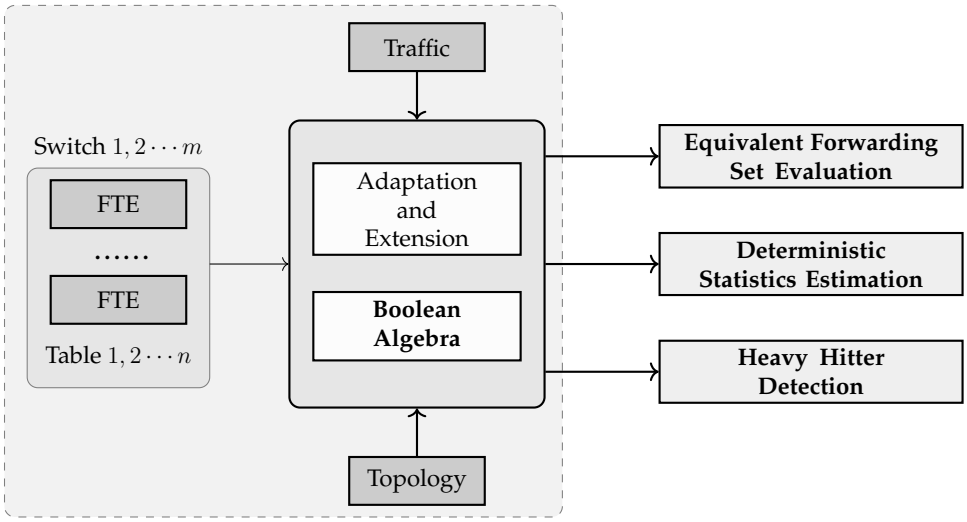


Figure 1.2: Research framework

In this thesis, Boolean algebra is chosen to express and explore the capability of SDN. Some adaptations and extensions are made on Boolean algebra to make it fully fit in with SDN forwarding pipeline. Unlike the existing research which only takes single table into consideration, multiple flow tables as well as the tables across multiple switches are investigated in this thesis. Moreover, real-time traffic and topology information are also integrated with FTE to assist an SDN controller to better understand and manage all the resources inside a software defined network. In this thesis, FTE, topology, and traffic statistics are considered as inputs, Boolean algebra with its extension is the major research method, and the

three applications become the outputs of Boolean algebra on these inputs.

1.6 Organisation of Thesis

The remainder of this thesis is outlined as Table 1.1. Chapter 2 presents a literature review of the forwarding pipelines' representations and applications in traditional networking as well as software defined networking. In the next three chapters, the applications of Boolean algebra on FTE have been fully discussed and researched. Specifically, they are equivalent forwarding set evaluation on multiple flow tables inside a single switch, non-invasive deterministic traffic statistics estimation of an individual switch and heavy hitter detection across multiple switches, respectively. Finally the content and outcome of this thesis have been summarised in chapter 6. Several potential research paths are also derived to benefit the future development of SDN.

Table 1.1: Organisation of Thesis

Introduction		Chapter 1
Background and Related Work	SDN Overview	Section 2.1
	FTE Overview	Section 2.2
	FTE Representation	Section 2.3
	FTE Application	Section 2.4
Contributions	Equivalent Forwarding Set Evaluation	Chapter 3
	Deterministic Statistics Estimation	Chapter 4
	Heavy Hitter Detection	Chapter 5
Conclusion		Chapter 6
Appendicies	Boolean Reasoning	Appendix A
	Boolean Function Operations on Match Fields	Appendix B
	Formula vs Truth-table Comparison	Appendix C
	Test Environment Setup	Appendix D
	Acronyms and Abbreviations	Appendix E

Chapter 2

Background and Related Work

The packet forwarding behaviour of a network relies on the rules residing in the networking element (e.g. switches and routers) to forward packets. This is true for both traditional networking and software defined networking, irrespective of a distributed or centralised network design. These rules vary in their structures and sizes, but they represent the same fundamental functionality: to match incoming packets against rules and determine its forwarding behaviour. All these rules in a network device constitute a packet forwarding pipeline, they mainly include: Access Control List (ACL) and IP routing entry in traditional networking as well as a more generic type of forwarding table entry (FTE) in SDN.

Upon surveying the existing work on network forwarding, FTE appears to be a generic form of ACL and IP routing entry in SDN. FTE is the core element that bears responsibilities of forwarding packets according to controllers' instructions. This chapter presents a literature review of the representation and application of forwarding pipeline. The chapter offers an extensive overview on the state-of-the-art advances in representations of the packet forwarding pipelines that contributes to the understanding of key features of a packet forwarding behaviour. This chapter reviews existing and up-to-date technical solutions, identifies their basic characteristics to derive the essential FTE attributes which must be covered by

an FTE representation.

The scope of this background and related work includes FTE representation and networking applications based on the analysis and manipulation of FTE. In the review of representation, the forwarding pipeline in traditional networking and SDN are inspected and characterised. Moreover, the various representations to analyse FTE as well as the rationale why they are chosen and their respective applicable scenarios are presented. In the review of applications, two types of applications are surveyed: equivalent forwarding table evaluation and traffic monitoring.

2.1 Software Defined Networking: Overview

With the introduction of SDN, the control plane of a network is moved to a single entity while in traditional networking, switches and routers are responsible for forwarding packets. This entity can be a physical single device such as a server or a set of logically centralised but physically distributed servers. A simplified architecture view of SDN is depicted in Fig. 2.1. The very core of a SDN enabled network is the controller which not only exercises direct control over all forwarding devices, but also responds to the requests from the application side. A controller communicates with the higher-level components, the applications, via the northbound interface. Similarly, it also communicates with the lower-level components, the network devices, via the southbound interface. The first standard southbound interface is OpenFlow which provides an industry-standard application programming interface (API) and protocol to program forwarding tables in switches.

While OpenFlow is the first and probably most well-known southbound interface, it is not the only one available for SDN. The traditional network management protocols such as NETCONF, SNMP or routing protocol such as BGP have been reused to configure the switches in SDN. Even though they cannot provide the same flexibility as OpenFlow, they

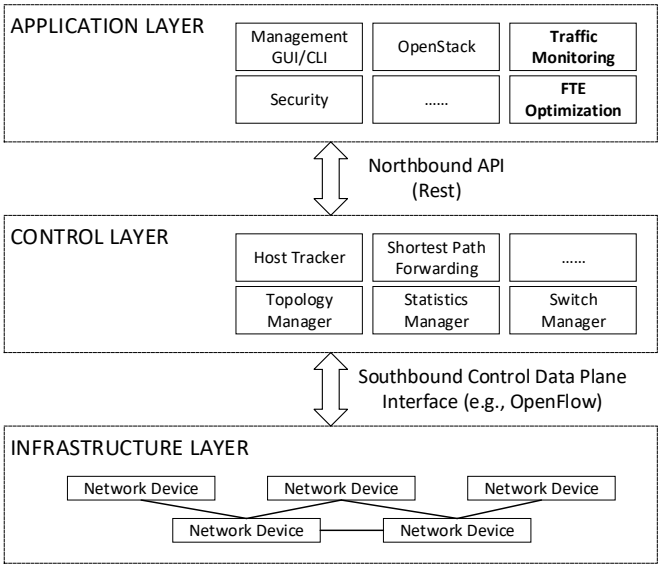


Figure 2.1: Simplified SDN Architecture

are still categorized as southbound APIs. Besides, there are some alternatives to OpenFlow which include Open vSwitch Database Management Protocol (OVSDB) [30], ForCES [31], OpFlex [32], etc. OVSDB manages Open vSwitch implementation following the same way as OpenFlow. It just adopts a non-OpenFlow protocol to programme OpenFlow switches. Similar to OpenFlow, both OpFlex and ForCES follow the policy-driven mechanism. OpFlex replaces the OpenFlow controller and OpenFlow interface with "Policy Authority" and "Policy Agent", respectively. However, the switch in OpFlex still relies on the FTE-alike flow tables to forward packets.

As a competing protocol to OpenFlow, ForCES is different from OpenFlow in many aspects, but they share the same design principle in forwarding models. ForCES manages the packet forwarding behaviour with an abstraction of Logical Functional Blocks (LFBs) which share the similar functionality as OpenFlow FTE. Both LFBs and OpenFlow tables consist of "Match-Action"-alike entries, i.e. FTE. In view of this, though most research in this thesis is exemplified through OpenFlow, the underlying

principles extend to non-OpenFlow protocols too.

2.2 Forwarding Pipeline: Overview

In networking, a pipeline is a chain of packet-processing entities (or filters) connected in a certain type of structure, where the output of one entity is the input of another one. For an individual device such as switch or router, the entities include the various chained forwarding tables, for example, ACL table, IP routing table in traditional networking and generic forwarding table in SDN. For a network, the entities include forwarding tables in all devices which are connected with a certain type of topology. It is also called ***packet forwarding pipeline*** because its major function is forwarding data packets between two devices.

2.2.1 Forwarding Tables in Traditional Networking and SDN

As illustrated in Fig. 2.2, a packet's forwarding path is determined by the forwarding tables in each element. The forwarding tables in traditional networking are usually statically configured or generated by locally running routing protocols while the tables in SDN switches can also be dynamically provisioned and updated by a centralised controller.

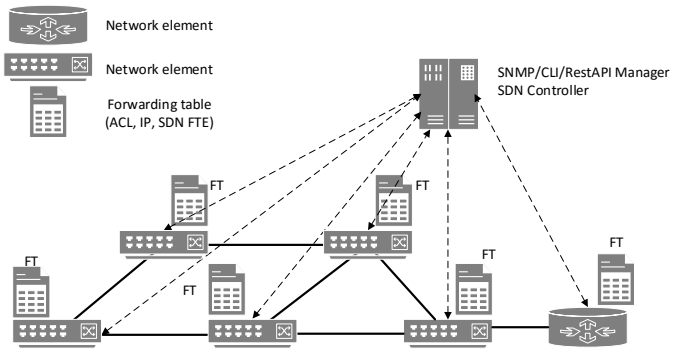


Figure 2.2: A network abstraction: forwarding tables

Traditional networking mainly relies on network protocols to perform packet forwarding. However, some applications, such as firewall and load balancing, need more flexible ways to dynamically regulate the network, which is unlikely to be specified by a protocol in advance. To meet this requirement, the concept of an ACL was introduced by Cisco to provide an alternative to manipulate the switch forwarding behaviour. Thus network management is partly achieved by ACL policy together with other pre-configured protocols: the incoming packets match against these ACLs and execute certain actions. This so called "policy-driven network" design is a primitive type of forwarding table in SDN [33]. An ACL can be considered as a special purpose filter in a forwarding pipeline which is designed to specify the access rights allowed or denied for all incoming packets.

Besides ACL table, IP routing table has also been widely used in a forwarding pipeline [34]. In a network, each networking device maintains a routing table which consists of a set of IP prefix entries and their associated egress interface(s). When an IP data packet reaches at a device, this packet's destination address will be matched against the routing table to find its egress interface(s) by matching against the longest IP prefix. Compared to ACL, IP routing table is a generic purpose filter and the most important components of a forwarding pipeline, it determines a packet's forwarding path in a network.

The core concept of SDN is that by creating a few careful abstractions based on ACL and IP routing table, the complexity of the underlying components is hidden and the new functionalities are easily developed via the programming interface between the network devices and the controller. One popular interface is OpenFlow, the de facto standard that gives access to the forwarding plane of a network. In OpenFlow, FTE has replaced the ACL and routing entry in traditional networking to manage the forwarding capabilities of the switches and routers. Packets are forwarded according to the path which is composed of these entries. For a pure software-defined network, the behaviours of all packets can be pre-

cisely controlled by manipulating the FTEs.

OpenFlow was proposed in year 2008 and it is still evolving [35], the mechanism and usage of FTE are not fully researched. The core idea of FTE is similar to the policies which have been widely used in ACL while the policy's scope and depth have been largely extended in OpenFlow. Since FTEs are not easily understood and manipulated as the network scales and grows in complexity, it is important to find a way to represent it.

The concept of FTE is central to this thesis's research framework (Fig. 1.2) because it directly determines the packet forwarding behaviour in a software defined network and it is also the key element to connect controller and switch. The two major perspectives of OpenFlow specification are FTE manipulations (installation, deletion, modification) and statistics updates [36]. This motivates the research on FTE representation and exploiting FTE to better manage a software defined network.

2.2.2 Past Work for Characterising Forwarding Behaviour

Previous work investigating forwarding behaviour can be classified into i) ACL, ii) IP routing entry, iii) FTE. All of them are the filters in a pipeline but their structure and functionalities vary. This section reviews the past work for all these three table entries and derives the four essential attributes for a successful forwarding pipeline representation.

2.2.2.1 Regulating Forwarding Behaviour by ACL

An ACL is composed of a sequence of rules to match against the packet to determine whether a specific action is performed or not. It is a special type of role-based access control (RBAC) [37]. A "minimal RBAC Model", RBACm, can be compared with an ACL mechanism, ACLg, where only groups are permitted as entries in the ACL [38]. ACL has been widely used in computer system, such as the user account authentication, firewall, etc. The prior research on ACL is mainly focused on the validation of these

access control policies and the optimization of their storage space.

Lampson introduced the formal notions of *subject* and *object* as well as an access matrix that mediated the access of subjects to objects [39].

Definition 2.2.1. Access matrix
An *access matrix* consists of a set of subjects $s \in S$, a set of objects $o \in O$, a set of operations $op \in OP$, and a function $ops(s, o) \subseteq OP$, which determines the operations that subject s can perform on object o.

Figure 2.3 shows a select host in NetB being granted permission to access NetA. All traffic sourced from Host B destined to NetA is permitted while all other sourced from NetB destined to NetA will be denied. Its corresponding access matrix is demonstrated in Table 2.1. The specified operations (*permit host 192.168.10.1*) are performed on all incoming packets which are the implicit objects in an access matrix.

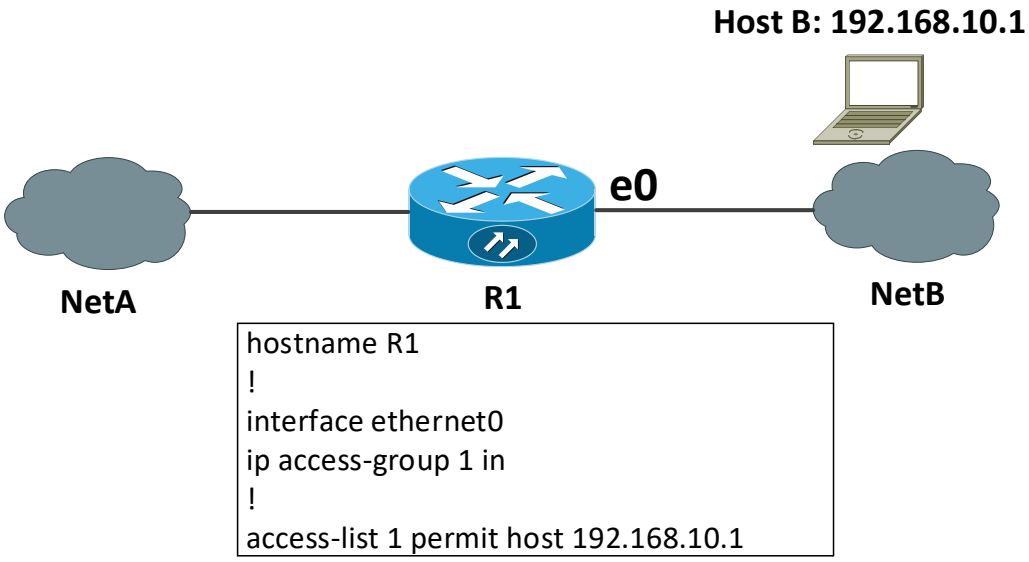


Figure 2.3: An ACL example

An access matrix can be easily transferred to the "match-action" form (MAF) which has been specified in OpenFlow specification [2]. For the case in Fig. 2.3, its equivalent MAF is illustrated in Table 2.2. An MAF

Table 2.1: Access matrix for the ACL example in Fig. 2.3

subject	object	operation
Interface ethernet0	Incoming packets	Permit host 192.168.10.1
*	*	Deny

does not include the field of object in access matrix because all ACLs share the same object: incoming packets. The action in MAF only contains two operations: permit and deny, all the rest information in an access matrix is transferred into the match fields in a MAF. The default action for all packets which are not explicitly permitted by ACLs is denial, this is why the second row is added in Table 2.2. The match field in a MAF is composed of an access matrix's subject and the negatives of the fields in operation. From the perspective of functionality, an ACL is equal to a generic forwarding table in which no constraints on match fields while only two type of actions are allowed: permit and deny.

Table 2.2: "Match-action" form for the ACL example in Fig. 2.3

Match Fields	Action
Ingress port = Ethernet 0 Destination IP Address = 192.168.10.1	PERMIT
*	DENY

2.2.2.2 Regulating Forwarding Behaviour by IP Routing Table

An IP routing table determines which specific route is selected for a given IP address. This entry represents the smallest subnet that contains the given IP address. A routing table captures two aspects: composition and relation [40]. Composition analyses the components of a routing table and their respective meaning; Relation interprets a routing table from different functional perspective. The structure of an IP routing table is illustrated in Fig. 2.4.

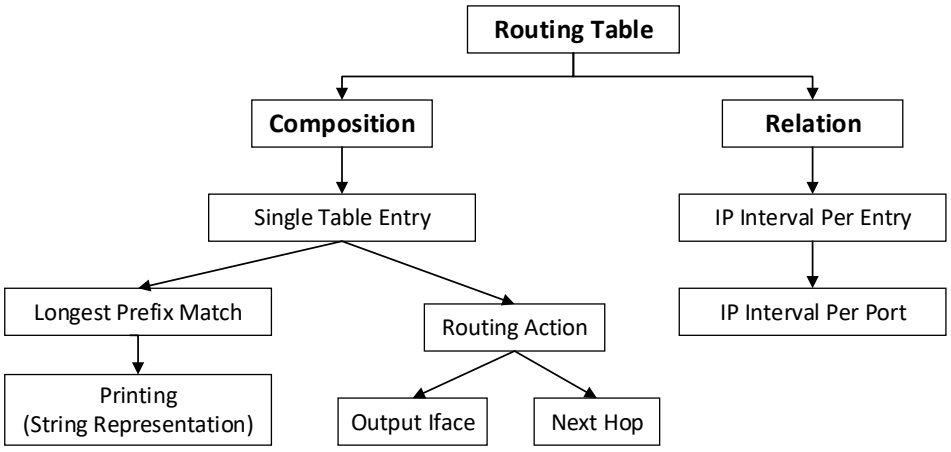


Figure 2.4: IP routing table structure

A routing table is a list of entries, each of them contains a longest prefix IP address and its corresponding routing action. Thus the relation between IP and action can also be understood as a function from IP (interval) to port (where the intervals don't overlap) or a function from port to IP (a set of IP space). Figure 2.5 is an adaptation of a necessary definition of a routing table semantics [40] in which the datatype *linord-helper* define the set relationship between two IP addresses (Lines 2-6). It is used to express the concept of longest prefix match (LPM) (Lines 13-16). For all IP packets which match the given routing table (Lines 8-11), they will be forwarded to the corresponding output interface or next-hop depending on their associated routing-actions (Lines 17-19).

2.2.2.3 Regulating Forwarding Behaviour by FTE

OpenFlow is a de facto interface between controller and switch. It provides an industry-standard application programming interface and protocol to program forwarding tables in switches. OpenFlow is managed by Open Networking Foundation (ONF), an organization dedicated to promoting and adoption of SDN. OpenFlow, of course, is evolving, and will continue to evolve. The latest OpenFlow switch specification is Version 1.5.0 (Pro-

```
 1: datatype ('a,'b) linord-helper = LinordHelper 'a 'b
 2: begin
 3:     definition linord-helper-less-eq1 a b ≡ [ case a of LinordHelper a1 a2 ⇒ case b of
    LinordHelper b1 b2 ⇒ (a1 < b1) ∨ ((a1 = b1) ∧ (a2 ≤ b2)) ]
 4:     definition a ≤ b ⟷ linord-helper-less-eq1 a b
 5:     definition (a ≠ b ∧ linord-helper-less-eq1 a b)
 6: end
 7: theory Routing-Table
 8: import ../IP-Address/Prefix-Match
 9:         ../IP-Addresses/IPv4 ../IP-Addresses/IPv6
10:         Linorder-Helper
11:         ../IP-Address/IP-Address-toString
12: begin
13:     record(overloaded) 'i routing-rule =
14:                         routing-match :: ('i::len) prex-match
15:                         metric :: nat
16:                         routing-action :: 'i routing-action
17:     record(overloaded) 'i routing-action =
18:                         output-iface :: string
19:                         next-hop :: 'i word optio
20: end
```

Figure 2.5: Routing table semantics

tocol version 0x06) [2] which was released on December 19, 2014 by ONF.

Since OpenFlow Version 1.1.0 [41], a flexible pipeline with multiple tables is exposed to control layer. It changes OpenFlow table processing significantly. As depicted in Fig. 2.6, packets are processed through a pipeline which consists of one or more tables, in each table multiple FTEs reside. The first table, labeled "Table 0" in Fig. 2.6, must be matched against all packets, and subsequently the packets might jump to any following table according to the pipeline instructions such as metadata which carries information between tables. Once a packet hits an OpenFlow entry, the instructions of this entry are applied immediately or accumulated in the action set associated with this packet and carried to the next processing table. The action set will be finally executed at the end of the pipeline and applied to the packet. Sometimes an FTE points to a group table which consists of a set of actions. According to the group types (for example, all, select, indirect and fast fail-over), either one or all actions will be executed.

The match fields and actions in FTE can be easily customised to imple-

ment the same forwarding behaviour as ACL and IP routing entry [42]. For an ACL, the match fields remain the same as FTE, only the ACL's default action "deny" is replaced by "drop". The conversion from an IP routing entry to an FTE is more straightforward, neither the match fields or actions need any changes. But not all match fields in traditional networking are explicitly indicated, for example, usually only the packets with specified VLAN id and destination MAC (DMAC) address will be sent to IP routing table for prefix searching. To maintain the same forwarding behaviour for an existing IP routing entry, the match fields of FTE will become the combination of "VLAN, DMAC" as well as the original IP prefix in routing table.

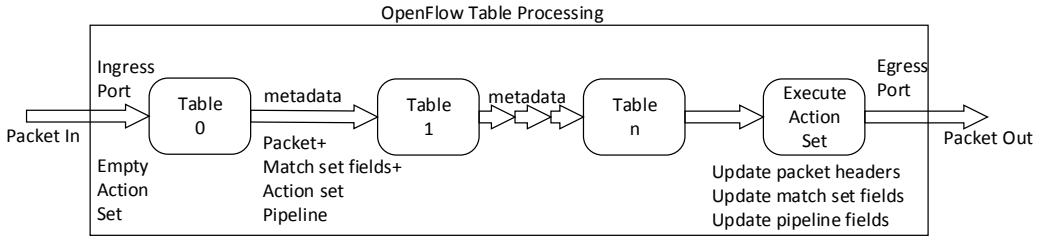


Figure 2.6: OpenFlow table processing

A flow table consists of multiple flow entries. Figure 2.7 illustrates the main components of an FTE: match fields, instructions, counters, timeouts and priority. An FTE is identified by match fields and priority: the match fields determine whether a packet can hit an FTE while the priority determine whether a packet has the chance to match against this FTE. Only the matching flow entry with the highest priority will be selected. A match field contains the well-known fields in an IP header (source MAC, destination MAC, source IP, destination IP, etc.) as well as the fields related to pipeline processing, for example, ingress port and metadata.

From the SDN forwarding table research in [43–45], there are four essential FTE attributes: i) Wildcard [W], ii) Priority [P], iii) Multi-table [M], iv) Topology [T].

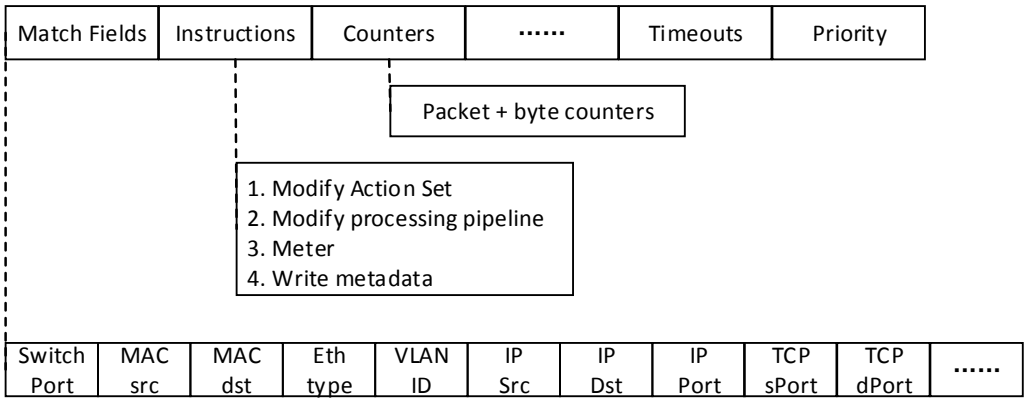


Figure 2.7: Inside FTE: OpenFlow packet matching and actions

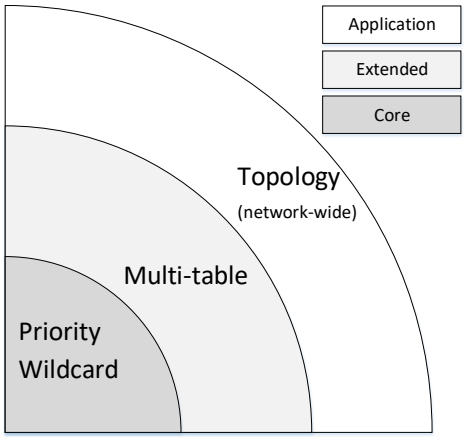


Figure 2.8: OpenFlow attributes

Among the above key attributes, wildcard and priority are only associated with a single table. Wildcard and priority inherited from ACL and a lot of research has been done on these attributes [46, 47]. As illustrated in Fig. 2.8, they are categorised as core attributes and must be covered by all representations. Multi-table is one of the most significant feature in OpenFlow to enrich dynamic configuration and placement of FTE. Multi-table is listed as extended attributes in Fig. 2.8 and it must be represented by any network-wide representation. Although topology is not an intrinsic attribute of FTE, it is still listed here as a crucial attribute because it plays

an important role in determining the network-wide packet forwarding behaviour.

2.2.3 Pipelines: a Common Denominator for Packet Forwarding

Inside a single forwarding element (switch or router), there might have multiple forwarding tables - each table serves a different purpose - to match their respective specified field(s) against the incoming packets. These tables construct a packet forwarding pipeline which is the common denominator for ACL and IP routing table in traditional networking as well as the generic forwarding tables in SDN.

Since IP routing table has only one match field - the destination IP address to match upon the incoming packets, it is also called a single dimensional forwarding pipeline [48]. The ACL, on the other hand, is a multiple dimensional pipeline because it matches against more than one known field simultaneously. However, as the name ACL suggests, it can only decide whether a packet is allowed to pass through a device by its predetermined action: permit or deny. The SDN paradigm uses a more generic forwarding pipeline which allows arbitrary combination of multiple fields and performs more complex actions.

Table 2.3 depicts their respective scopes of match fields and actions. From the perspective of "match fields", the IP routing table only has one dimension which is the IP prefix while the rest two types have arbitrary combinations of the fields in a packet header. From the perspective of "actions", IP routing table always directs to the next-hop and associated with one egress interface.

Though the structure and functionalities of forwarding pipelines have changed significantly from traditional networking to SDN, the underlying hardware almost remains the same. Most SDN switches on the market share the same hardware as before. Figure 2.9 and 2.10 illustrate the

Table 2.3: Forwarding table comparison

Forwarding Table Type	Match	Action
ACL	multiple dimensions	permit/deny
IP routing table	single dimension	egress interface
FTE	multiple dimensions	arbitrary actions

reference forwarding pipeline of a typical ASIC (application–specific integrated circuit) based switch in traditional networking and SDN, respectively.

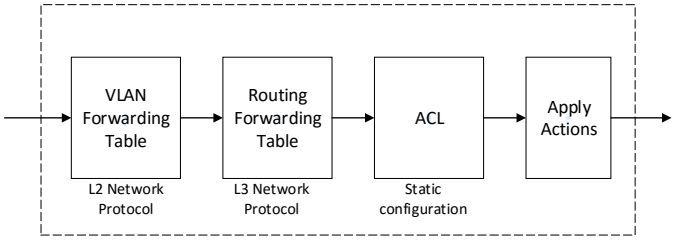


Figure 2.9: Switch-level pipeline in a traditional network

In traditional networking, each table has fixed width and is assigned the corresponding predetermined functionality, for example, VLAN validation in the first table, IP address prefix matching in the second table and ACL filtering in the third table, etc. However, in SDN, these table can be customised and they are not restricted to the predetermined actions. Hence, the ACL and IP routing table can also be considered as a special type of forwarding pipeline. A forwarding pipeline is composed of one or multiple forwarding tables which contain forwarding table entries to specify the match fields and their associated actions.

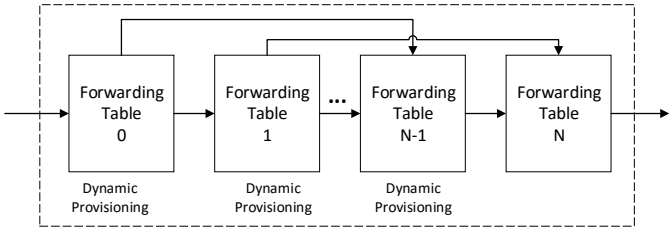


Figure 2.10: Switch-level pipeline in SDN

2.3 FTE Representation

2.3.1 Past Work for FTE Representation

Even though the OpenFlow standard is still evolving, various approaches have been proposed to analyse and manipulate the FTE in OpenFlow. A summary of the literature of FTE representation is presented in Table 2.4 and 2.5. They use either simple logical connectives to explore FTE manipulations [3, 4] or advanced methodologies to abstract and represent FTE. They are built upon single logic or a combination of multiple logics. The prior single-logic based representations include algebra [5,6,49], set theory [7, 8], first-order logic based model checking [9], temporal logic [10–12], and higher-order type theory [13, 27]. Some representations adopt more than one logic which includes Z [6, 49] and multiple-logics based model checking [14]. The Z representation is developed from typed first-order predicate logic and Zermelo-Fraenkel set theory.

Some research in Table 2.4 and 2.5 adopts the data structure based representation such as brick, tree and graph [15–20,22,50,51]. They will not be used in this thesis because they usually aim to resolve a specific problem rather than providing a generic solution. For example, the RuleBricks [23] provides high availability (HA) policies to OpenFlow forwarding pipeline. This thesis intend to find a representation which is not particular to any application. But they demonstrate that the low-level representations are easily applicable to the real networking applications.

Table 2.4: Summary of FTE manipulation

Project and Reference	Technique	Attrs*	Applications
FlowChecker [10]	Binary decision diagrams, Computational Tree Logic	[WP]	Identify misconfiguration
Frenetic [7]	Set theory	[WPT]	Application oriented network programming language over controller
DIFANE [16]	Rule partition	[WPT]	Scalability
NICE [14]	Propositional logic, temporal logic	[WPT]	Test and verification of OpenFlow applications
Slicing abstraction [11]	Computation Tree Logic	[T]	Slicing and Isolation
NetCore [8]	Declarative language, Domain specific languages, Set-theoretic operations	[WPT]	A compiler, transformation from application-level policies to switch-level policies transformer
Formal Specifications [6, 49]	Algebra of communicating shared resources	N/A	Formalisation and verification of SDN framework
FlowAdapter [15]	Tree composition/de-composition	[M]	Multi-table processing on legacy hardware
OpenFlow rules interaction [3]	Logic connective	[WP]	Generic analysis, management and optimization of FTE
Featherweight OpenFlow [13, 27]	Coq [52], Domain-specific languages	[WPT]	SDN controller verification
FLOVER [9]	First order logic, Logical connective	[WP]	Verification of security attributes in OpenFlow

* Attrs: W - Wildcard; P - Priority; M - Multi-table; T - Topology

Table 2.5: Summary of FTE manipulation - continued

Project and Reference	Technique	Attrs*	Applications
Automated Synthesis of Controller [12]	Linear temporal logic	N/A	Action-based synthesis of SDN
"One Big Switch" [22]	Rectangular representation and selection	[WPT]	Rule placement (Space, time, resource)
Palette [17]	Rule Decomposition, Graph-based model	[PT]	Rule decomposition, distribution and placement
vCRIB [20]	Tree based model, partition	[WPT]	Traffic-optimal rule placement
CAP for Networks [50]	CAP (consistency, Availability, Partition) theorem	[WT]	Trade-off between policy enforcement and network connectivity
Maple [21]	Algorithmic Policies	[WPT]	Simplify SDN Programming
RuleBricks [23]	Brick based model	[WP]	OpenFlow Failure-planning
CAB [19]	Tree decision model, partition	[WP]	Flow setup efficiency improvement
NetKAT [5,53]	Coalgebraic theory, Kleene algebra, Brzozowski derivative	[PT]	All-pair connectivity, Loop-freedom, Translation validation
Routing for Efficiency [54]	Integer linear programming mode	[T]	Maximise traffic satisfaction
EAR [18]	Graph-based model	[PT]	Energy-aware Routing
CacheFlow [4]	Graph theory, Recursive theory	[WP]	Caching system for SDN, FTE abstraction
Generalised FTE Optimisation [51]	Tree-based model, Tagging approach	[PM]	FTE compression

* Attrs: W - Wildcard; P - Priority; M - Multi-table; T - Topology

Logical connective representation mainly utilises logical operators to express the relation and interactions between FTEs. The relations that can be in place among match fields and instruction sets are first analysed in [3]. Based on potential relation combinations, they define five FTE interaction types: *Disjoint*, *Exact match*, *Subset*, *Superset*, and *Correlated*. More details on Boolean operations are further explained in Appendix B. They will be extended with the help of Boolean algebra and applied into all the three applications which will be further discussed in the next three chapters.

Algebra representation is good at describing the attributes and reasoning of a structure or program. The simplest algebra is ***Boolean algebra*** [55] in which the values of the variables are the truth values *true(1)* and *false(0)*. Another branch of well-studied algebra representation pioneered by E.F. Codd called relational algebra [56], was proposed to model the data stored in relational databases. ***Kleene algebra*** [57], partly built on relational algebra, focuses on the semantics of a program which can be expressed as an idempotent semiring. Another network programming language which builds on top of Kleene algebra, NetKAT [5,53], has demonstrated its capability of representation for the attributes such as priority and topology, but there is no evidence to show that it can also be used to represent multi-table in SDN.

Set theory (Frenetic, [7]) is concerned with the concept of sets. It studies the well-determined collections of objects. However, its strength lies in descriptiveness rather than manipulation, thus it is unlikely to fulfil the desired functionalities related to FTE manipulation, for example, removing the priority in a single table, combining multiple tables.

Symbolic logic uses symbols and variables to express logical ideas. It is by far the simplest kind of logic. Variants of symbolic logic includes ***propositional logic*** [58], ***predicate logic*** [59] and ***temporal logic*** [60]. Among these three logics, propositional logic has no quantifiers while the other two have. The quantifiers are quite useful to express the uncertainties in a forwarding pipeline, such as load balancing and failover [14], in both scen-

arios the same packet might be forwarded to different paths. Propositional logic studies the indivisible statements. It assumes every statement can be interpreted as true or false and then produces more complex statements in which truth-value depends on the truth-values of the simpler statements. The symbols or words used to connect two statements are logical connectives which include binary connectives such as *"or"*, *"and"* as well as unary connectives such as *"negation"*. It has been proven useful in modelling network forwarding pipelines where each entry is interpreted as a logical expression of the conditions to trigger their associated actions [58].

Since propositional logic is not able to represent the relationship between propositions, a more powerful logic, predicate logic, is studied [59]. Predicate logic is an extension of propositional logic and more expressive. It uses quantified variables such as existential $\exists$ ("there exists") and universal $\forall$ ("for all") over objects to define the scope of the statements. Predicate logic is a generic term of ***higher-order logic***.

These logics can describe functional relationships and statements about "for all" objects or about "for some" objects but their quantifiers vary. First-order logic quantifies over individuals of the domain of discourse. Higher-order logic is distinguished from first-order logic by additional quantifiers. As illustrated in Fig. 2.11, higher-order logic quantifies over sets or sets of sets while first-order logic has the quantifiers of non-nested sets.

Modal logic extends the classical proposition logic and predicate logic to allow the quantifiers to express modality, for example, "necessarily" and "possibly" [60].

Compared to logical connectives and Boolean algebra, the aforementioned logics in Fig. 2.11 are high-level abstraction which indicates that they are better at expressing the connections among FTEs, for example, the priority and topology. However, the low-level FTE attributes such as wildcard and "goto table" actions are easily overlooked by these logics.

In this thesis, Boolean algebra is selected as the technology to represent

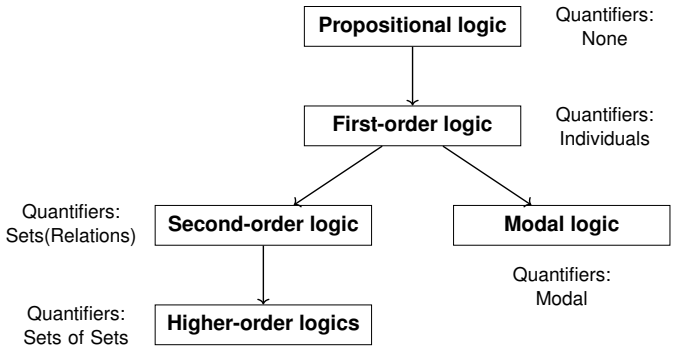


Figure 2.11: Quantifiers of logic

FTE, more justifications will be given in the next subsection.

2.3.2 Boolean Algebra for FTE Representation

In mathematics and mathematical logic, Boolean algebra is a branch of algebra in which the values of the variables are the truth values "true" and "false", usually denoted 1 and 0, respectively. Boolean algebra is naturally suitable to represent SDN forwarding pipeline especially on the applications such as the equivalent forwarding set evaluation and traffic monitoring. In the former scenario, the question turns to the Boolean function on the match fields for the same action. In the latter scenario, the question turns to find all supersets/subsets and retrieve their individual statistics for a given flowset.

To fit the Boolean algebra to SDN FTE, two challenges must be addressed: wildcarded logical operations and deprioritisation. Although in traditional networking, wildcard is only supported on lower bits (at the rear) of an IP address or MAC address to facilitate the longest prefix search, it may locate at any position in SDN FTE. Thus the Boolean algebra on FTE must allow the wildcard be positioned anywhere and accepted by all Boolean operations.

In this thesis, "wildcard" stands for "wildcard mask" which comes from CISIO. It is a mask of bits that indicates which parts of IP addresses

are available for examination. For example, a wildcard mask 0.0.0.3 represents that the last two bits are not available for examination. A packet with an IP address which ends with bit 0 or 1 will have the same examining result. A wildcard mask is continuous and always indicates the last certain number of bits are not available for examination. Thus a wildcard mask in a truth table has the following format: 1000011111****, 000111**, 110000* in which the number of asterisk is flexible while they always position at the rear of a Booelan value. However, in FTE, a more flexible format of wildcard is expected in a truth table, the wildcard (*) should be positioned anywhere (not only in the rear of a value), such as 10000**111****, 10000**0011, ***111000.

Another challenge is introduced by a FTE's unique attribute: "priority". Due to the existence of priority in a forwarding table, most FTEs' match fields cannot reveal their actual matching scope. After appropriate adaptations and extensions, Boolean algebra extends the existing SDN application into multiple flow tables across multiple switches.

Besides the intrinsic FTE attributes wildcard and priority, Boolean algebra is also capable of representing another two attributes: multi-table and topology. The Boolean operations (for example, logical AND ($\wedge$) and OR ($\vee$)) between any two forwarding tables or even two switches can be converted to the operations between the FTEs in their respective tables or switches.

Besides Boolean algebra, other mathematical logical methods have also been examined to investigate the possibility of representing FTE and exploiting their applications. It is found that propositional logic and some lower-order logic are also good at expressing SDN forwarding pipeline, but the abstraction in the process of FTE representation makes these approaches less applicable in practical scenarios because some low-level information has already lost. For example, the relative positions of multiple forwarding table entries and the connections among multiple tables are hard to be represented in these logics.

Among all the formal semantics, NetKAT represents single table very well and maintains most essential low-level information. However, the two challenges on Boolean algebra exist for NetKAT as well, which means the similar adaptations and extensions on Boolean algebra are also required for NetKAT. Moreover, its initial design does not cover the multi-table attribute, which further limited its applicable scope.

In a word, as a low-level representation, Boolean algebra is unlikely to reveal the whole picture of network-wide forwarding pipeline, but it is easily extended to support the latest SDN core forwarding attributes such as priority and multi-table. In the following three chapters, the flexibilities, manipulability and applicability of Boolean algebra have been well proven.

2.4 Applications Based on FTE Representation

Based on the analysis of FTE, various applications have been developed. They can be mainly categorised into two types from functionality perspective: FTE placement and Traffic monitoring. FTE placement mainly covers the following aspects: i) FTE manipulation [15, 61–63]; ii) consistent updates over multiple switches [64–68]; iii) placement optimisation [69]. All these three placement applications rely on the correct conversion among the different forms of tables across multiple switches. The works on traffic monitoring include traffic estimation [70], traffic anomaly detection [71], verification of forwarding tables [72], etc.

2.4.1 Optimising FTE

Prior works on FTE's placement optimisation can be further divided into two categories: single switch flow table optimisation [19, 23, 73–76] and network-wide flow table optimisation [16–18, 20–22, 50, 51, 54]. The optimisation of single table usually focuses on compression which heavily relies

on the mechanism “merge” and “ split”. The new installed rules are firstly merged with each other and then split according to the hardware table structure and size. Then the neighbouring or function-alike flow entries can be further merged to save table space. Another optimisation approach is to deploy FTEs according to their different access frequencies, for example, deploying the most frequently accessed FTE in high-speed cache and the relatively unpopular FTEs in low-speed cache [73]. In the scenario of FTE compression, a number of practical advances to increase the applicability of hardware resilience via forwarding table compression algorithm and compression-aware routing have been discussed [77]. A method to verify whether redundant rules exist and the way to avoid them during incremental deployment are also proposed and verified [76]. A major drawback of this “merge” and “ split” approach is that it fails to explore FTE’s multi-table capability. Some researchers notice the benefit of multiple flow tables and they perform partitioning and compression by distributing forwarding rules into different tables according to their respective capabilities [74]. Network-wide optimisation is usually achieved by “combination” and ”distribution”. In a given interval, all requests from different applications will be combined as a single composite request. Then they are reorganised and distributed to the devices. During this process, the exact path specified by the original application might be changed. It enriches the freedom to adjust the FTE placement across all applications, however, the achievements of some higher-level goals such as traffic engineering cannot be guaranteed.

A core question for any deployment of multi-table FTE is how to verify the equivalence between the table(s) to be provisioned and the table(s) deployed in the switches. Particular attention about multi-table evaluation and deployment should be paid to the research in [78] and [15]. The former proposed a reconfigurable match tables (RMT) model, which allows the forwarding plane to be modified by reorganizing the tables (for example, adding new match fields and reconfiguring IP lookup tables) [78]. The

latter proposed a middle layer to convert flow rules (entries) from the controller to switch hardware flow table pipeline, in which the transformation from one single policy to a multi-table rule is usually required so that the rules can be fitted into different types of hardware [15]. Both works are concerned with the conversion between different types of forwarding tables, which motivates the first contribution of this thesis , i.e., guaranteeing that the conversion between a single table and multi-table is equivalent in terms of networking functionality.

However, it is found that the OTN (the conversion of a One-stage flow table into N-stage flow tables) proposed in [15] is NOT always correct. Its core idea is to fill the corresponding match field value of a one-stage flow entry to multi-stage entries based on their containing match field types, as illustrated in Fig. 2.12. In this conversion, the original single table (Table 1) has been converted into a multi-table structure (Table 2 and Table 3). Take the first entry in Table 1 as an example, the original match fields {A, B} are decomposed and distributed into {A} in Table 2 and {B} in Table 3, respectively. However, closer analysis reveals that packets matching either {A, B} or {C, D} will behave the same while the packets with {A, D} or {C, B} will not. The packets with attributes {A, D} will NOT match any entries in the single table and will be dropped by default; however, in the converted multi-table, the same packets can match the first entry in Table 2 and then the second entry in Table 3 sequentially, which means the matching packets will egress out of port 2. For the same packets, the forwarding behaviour will not be the same, which means the single table is not equivalent to the multi-table. Thus, a systematic approach to evaluate the equivalence of any two given forwarding sets becomes a challenge for all forwarding table manipulations.

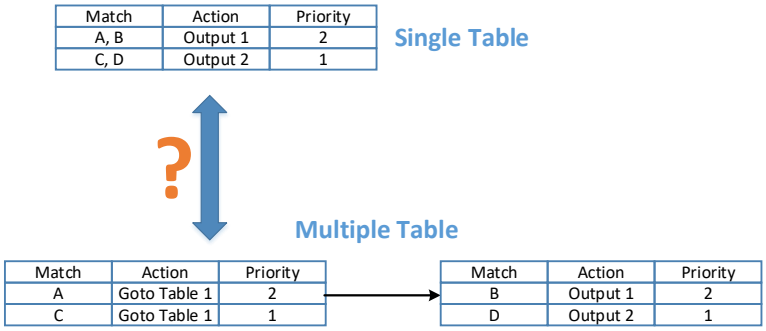


Figure 2.12: Match fields distribution of OTN conversion

2.4.2 Traffic Monitoring

The most widely adopted traffic monitoring approaches are either based on packet sampling or sketch-based measurements. Sampling-based monitoring solutions mainly include NetFlow and sFlow. NetFlow [79] is a feature on Cisco routers for collecting IP network traffic with predetermined rules. sFlow [80], short for "sampled flow", provides a means for collecting information of truncated packets, together with the interface statistics. Unlike NetFlow, sFlow offers greater scalability and reports OSI layer two to layer seven information on network traffic in detail [81] but it consumes more resources (e.g., CPU, memory, and bandwidth). A high sampling rate generates too much information (costly to store), while a lower sampling rate may result in heavy-hitter flows going undetected.

Compared with sampling-based monitoring, sketch-based approach can process millions of streams in a short time and with low overheads. It is a probabilistic summary of data streams within a compact data structure. The approach builds forecast models on top of sketches which represent the past traffic patterns. Sketch-based approach adopts an unique hash function and associates multi-dimensional tables to data streams for storing summarised data, which requires customisation of existing switch ASIC. This is why most of the existing works are only verified by simulation and implemented on field-programmable gate array (FPGA) [82, 83].

To fit the afore-mentioned approaches into the realistic networking monitoring scenario, ProgME (Programmable Network MEasurement) presents a framework to measure a flowset defined according to application requirement [84]. A statistics query is processed by a query answering engine which maps a flowset to unique flows whose statistics can be retrieved directly from hardware. Even though the scenarios of these traffic monitoring research vary, they share the same principle: to measure an arbitrary set of flows' traffic based on the ready statistics.

No matter what kind of approaches and frameworks are chosen, all the existing SDN traffic monitoring solutions use the same way to get the statistics. They install the application specified rules and then retrieve their statistics [83, 85–88]. The benefit is that the controller can customise the FTE rules for flexible traffic monitoring, for example, the controller is able to update the FTE rules to monitor the suspected malicious traffic dynamically. However, it is difficult to avoid the interference on the active traffic because the behaviour of all the packets matching with these monitoring FTEs will be altered.

2.5 Summary

This chapter reviews the recent literatures on the representation and application of packet forwarding pipeline. The review indicates that Boolean algebra is suitable for the low-level representation of forwarding table entries and more importantly, it also facilitates the applications such as equivalent forwarding set evaluation and traffic monitoring. From the next chapter, the SDN applications based on Boolean algebra will be further discussed: equivalent forwarding set evaluation (next chapter, chapter 3); deterministic statistics estimation (chapter 4) and heavy hitter detection (chapter 5).

Chapter 3

Equivalent Forwarding Set Evaluation

Network devices rely on forwarding rules to convey packets. Individual devices use these rules to construct a forwarding set to determine the behaviour of incoming packets. Forwarding rules appear in the form of access control list and IP routing entries in traditional networking while it is implemented as a generic forwarding table in SDN. In SDN, the design and structure of forwarding tables have progressively become more complicated to support evolving network applications. According to OpenFlow Specification 1.5 [2], these rules are constructed either in the form of a single table or multiple linked tables. The multi-table approach advocated by the OpenFlow specification has been widely adopted by commodity switch manufacturers, while the software controlling the switches use a single table for better manageability. A fundamental question is how to reconcile the forwarding behaviour of the multiple tables and single table. By converting any given arbitrary OpenFlow table into a uniform representation called *equivalent forwarding set* (EFS), this chapter formalises and implements the process of evaluating equivalence between two forwarding sets in terms of networking function. This research will facilitate the flow table management in controller where an equivalent table is usually

maintained as well as the flow table deployment in switches where multiple flow tables are more often chosen to adapt to switch's various forwarding pipeline design.

3.1 Research Question

A physical network is composed of multiple nodes and the links to interconnect them. Individual network nodes use forwarding rules as inputs to construct a forwarding set, and this forwarding set determines the movement of incoming packets passing through the node. These nodes may use a single table or linked multiple tables to implement forwarding rules.

In SDN, match fields in forwarding tables are dynamically customised by an intelligent, centralised controller. Moreover, SDN also allows these tables to be chained together to process the incoming packets in a more sophisticated way. Figure 3.1 illustrates the structure of multiple flow table in OpenFlow specification. A multi-table forwarding set example with three linked tables is presented in Fig. 3.2. These tables collectively form a forwarding set to match against all incoming packets and execute associated actions. Here the forwarding set refers to the rules within a single node spanning one or multiple tables. In this figure, the packets with fields "A" and "B" will be sent to Table 1 and Table 2 for further processing, respectively. The actions in "Table 0" contain "Goto Table" attribute which indicates the matched packets of this forwarding entry will be passed to next table for further processing. The row "A, Goto Table 1, 200" in "Table 0" is a flow table entry and it is linked to "Table 1", "Table 1" is also referred as this entry's associated table and a link is added between them to represent their association relationship.

The motivation of this research lies in the fact that forwarding sets are interpreted in various forms (single table, for example, fixed-width fixed-length multiple-table, dynamic-reconfigurable multiple-table) in different networking elements [89]. In an SDN controller, a logical single table is

usually maintained because the forwarding sets in a server has far less limitations compared to the rules in a hardware switch. The table structure and size in hardware switches vary significantly due to different hardware specification. For any transformation of a forwarding set among those various networking elements, the functional equivalence must be guaranteed. This research aims to find a solution to evaluate the equivalence of any two forwarding sets. The generic principle is to convert different types of forwarding sets into a canonical form and then compare each other with the help of Boolean reasoning.

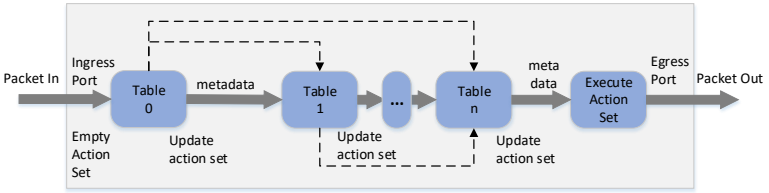


Figure 3.1: Multi-table forwarding set specified by OpenFlow

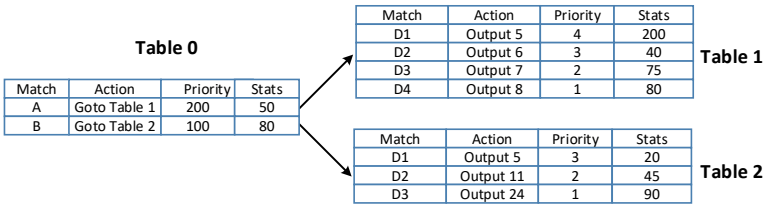


Figure 3.2: Multi-table forwarding set example

Since OpenFlow is the de-facto standard for SDN, the solution to the afore-mentioned question will be exemplified using OpenFlow specification 1.5. However, the approach presented in this chapter targets a generic conversion method which is not limited to OpenFlow [90].

Multiple flow table (MFT) is one of the most significant enhancements of OpenFlow 1.1, which adds power and flexibility to an OpenFlow switch [91]. In recent years, some works have noticed the benefits of MFT and applied it into SDN deployment, which requires a forwarding set to be

presented in different forms to adapt to the various networking applications. However, MFT introduces additional complexity and challenges to the software responsible for managing these tables. Thus a systematic way to evaluate the functional equality between two forwarding sets becomes indispensable.

Match fields in SDN can be naturally expressed as Boolean functions. Intuitively, the research work on Boolean logic is also applicable to networking forwarding set. The related fundamental concepts, equations and theories were explained very thoroughly in [92]. The definitions and theorems which are used in this chapter are listed in Appendix A and C.

3.2 EFS: Towards a Uniform Table Structure

Definition 3.2.1 (Equivalent Forwarding Set (EFS))**.** For any two given forwarding sets F_g and F_h, no matter what their structures and forms are, if they perform the exact same operations for all incoming packets $p_{in} \in \mathbb{P}$, they are considered as the equivalent forwarding sets.

In Definition 3.2.1, the operations include any modification on the packets themselves and their output interface(s), i.e., $(P_{out}(g) = P_{out}(h)) \wedge (P_{if}(g) = P_{if}(h))$ where P_{out} represents the egress packets and P_{if} represents the output interface(s). The typical modifications of IP packets include MAC address replacement and VLAN tagging, etc. The interfaces include physical egress ports (output (port) 1 & 2 in Table 3, Fig. 2.12) as well as the next hop in a logical pipeline which has been widely used in networking virtualisation. According to Definition 3.2.1, the incoming packets will have the same modifications and egress port(s) on a switch with two EFSs.

Fig. 3.3 illustrates two equivalent forwarding sets in the form of a single table and MFT. The multi-table forwarding set consists of three tables which are labelled "Table 0", "Table 1" and "Table 2", respectively. In this multi-table structure, the actions in "Table 0" contains "goto table"

attributes which indicate that the matched packets of the forwarding entry in "Table 0" will be passed to another table for further processing. In this case, this specific table becomes the entry's associated table and a link is added between the entry and the table. It can be easily verified that all the incoming packets will be forwarded to the same destinations without any modification in these two EFSs.

Using Definition 3.2.1, an MFT can be replaced by a single equivalent table and vice versa, i.e., these two sets are interchangeable.

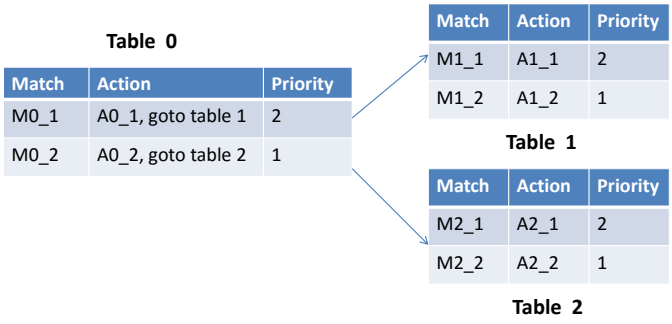


Table 0

Match	Action	Priority
M0_1	A0_1, goto table 1	2
M0_2	A0_2, goto table 2	1

Table 1

Match	Action	Priority
M1_1	A1_1	2
M1_2	A1_2	1

Table 2

Match	Action	Priority
M2_1	A2_1	2
M2_2	A2_2	1

Table 3: Equivalent Single Table

Match	Action	Priority
M0_1, M1_1	A0_1, A1_1	4
M0_1, M1_2	A0_1, A1_2	3
M0_2, M2_1	A0_2, A2_1	2
M0_2, M2_2	A0_2, A2_2	1

Figure 3.3: Equivalent forwarding example

3.3 Equivalent Forwarding Set Conversion

In this section, the conversion of a forwarding set into an equivalent uniform single-table is investigated to facilitate the comparison of two forwarding sets. Two approaches are proposed to achieve this: (i) *match-field oriented approach (MFA)*, which builds indexing on "match-fields"; and (ii) *action oriented approach (ACA)*, which constructs its forms on "actions".

The rest of this section presents these two conversion approaches with the same multi-table forwarding example in Fig. 3.4.

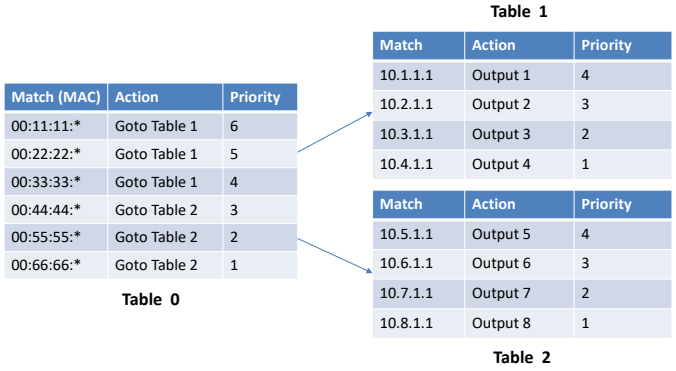


Figure 3.4: Multi-table forwarding example

3.3.1 MFA Conversion

In the match field oriented approach, a forwarding entry in one table and its associated tables will generate a new composite entry. This new entry preserves the functionality of the original entries. Thus, the link between multiple tables can be removed once the original entry and the link are replaced by an "equivalent" composite entry. Algorithm 1 describes the conversion process which removes one table per step i until the a composite single table is achieved.

Taking the MFT set in Fig. 3.4 as an example, Table 0 (with its entries) has two associated tables: Table 1 and Table 2. Each entry in Table 0 generates a new entry for every entry in Table 1 or 2. Thus, the equivalent single table will have 24 entries which are far more than the sum of the original three tables. In the worst case, for a MFT with N entries and t tables, the number of entries in its equivalent single table is in the order of $O(N^t)$. The worst case is easily constructed with a chained MFT structure in which every table, except the last one, has an associated table.

Once a multi-table forwarding set has been converted into a single-

Algorithm 1 Match fields oriented table join

1: $T \leftarrow$ input: An array of all the tables in a switch
2: $e \leftarrow$ input: Forwarding table entries in table T
3: $S \leftarrow$ output: Equivalent Single Table
4: *MAX_PRI* $\leftarrow$ constant: Maximum value of priority
5: *.cnt* $\leftarrow$ count of an array (T)
6: *.m, .act, .pri* $\leftarrow$ match fields, actions and priority of a flow (el, ek, e)
7: **procedure** $Multi\_table\_join(T)$
8: $n \leftarrow T.cnt$, $e \leftarrow$ new empty flow
9: **for** $i = 0$ *to* $n - 1$ **do**
10: **for** $ek \in T_i$ **do**
11: **if** $ek.act$ **has** $goto\_table\_id = n - 1$ **then**
12: **for** $el \in T_{n-1}$ **do**
13: $e.m \leftarrow ek.m \wedge el.m$
14: $e.act \leftarrow ek.act \cup el.act$
15: remove $goto\_table\_id$ in $e.act$
16: $e.pri \leftarrow ek.pri + el.pri/MAX\_PRI$
17: insert e into T_i
18: **end for**
19: remove ek from T_i
20: Convert all priorities to integer in T_i
21: **end if**
22: **end for**
23: **end for**
24: remove T_{n-1} from T
25: **if** $T.cnt = 1$ **then**
26: $S \leftarrow T$
27: **else**
28: $Multi\_table\_join(T)$
29: **end if**
30: **end procedure**

Note:
A table (T) consists of multiple flow entries (el, ek, e). Every flow entry contains multiple properties (match field ($.m$), action ($.act$) and priority ($.pri$)).
The operator $\wedge$ in Line 13 is a logical "AND"-alike operation. The result of a $\wedge$ operation on two match files yield a new match field which means the incoming packet must match both of them.

table set, the evaluation of the equivalence of these two sets is a Boolean comparison of the entries between two individual tables. Algorithm 2 presents a comparison of two tables based on a strategy in which all items of one table will be successively eliminated.

For two given tables T_1 and T_2, every item i_1 in T_1 will be compared with all the items in T_2 in the decreasing order of priority. If two items share same match fields while their actions differ, it can be concluded that these two tables are not equivalent (Algorithm 2, Line 8-11). For those two items with the same action, the common match fields will be removed from both items (Algorithm 2, Line 19-28). This process is repeated until the match fields in item i_1 yields an empty set, which means i_1 is subsumed by T_2. Otherwise, these two sets are not equivalent (Algorithm 2, Line 30-31). Since all items in the forwarding sets are sorted according to decreasing priority, the scope of match fields must be either independent or increasingly expanded. This order will also accelerate the process of the successive elimination comparison between two tables (Fig. 3.5).

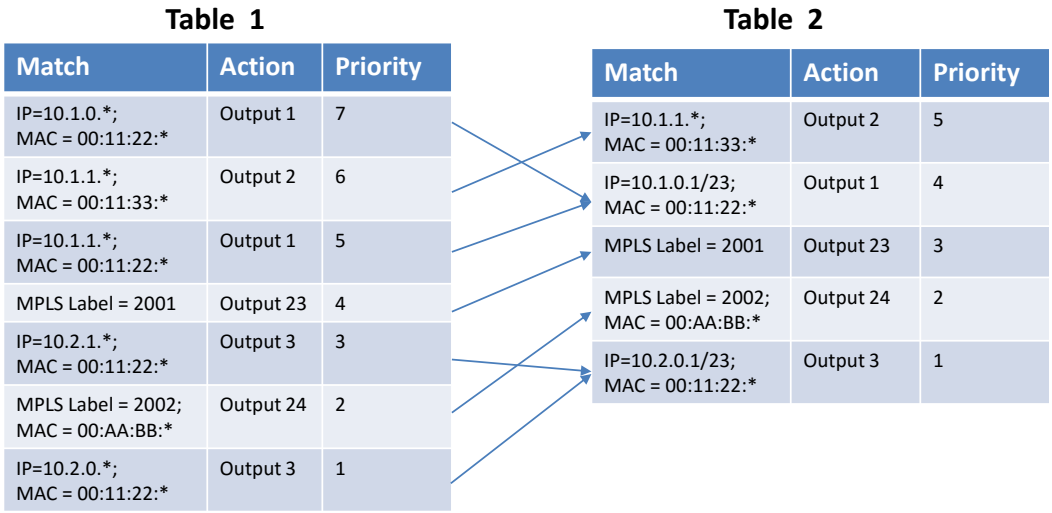


Figure 3.5: Successive elimination comparison

Algorithm 2 Two-tables-comparison

```
1: T_1, T_2 ← input: Arrays of two single tables
2: True, False ← output: Equivalence of T_1, T_2
3: .cnt ← count of an array (T_1, T_2)
4: .m, .act ← properties: match fields and actions of a flow entry
5: procedure Table_comparison(T_1, T_2)
6:     for i = 0 to T_1.cnt − 1 do
7:         for j = 0 to T_2.cnt − 1 do
8:             if T_1[i].act ≠ T_2[j].act then
9:                 if T_1[i].m ∧ T_2[j].m ≠ ∅ then
10:                    return False
11:                end if
12:            else
13:                if T_1[i].m = T_2[j].m then
14:                    if (i = T_1.cnt − 1)&(T_2.cnt = 1) then
15:                        return True
16:                    end if
17:                    remove flow entry j from T_2
18:                    break
19:                else if T_1[i].m ⊂ T_2[j].m then
20:                    T_2[j].m ← T_2[j].m ∧ (¬T_1[i].m)
21:                    break
22:                else if T_2[j].m ⊂ T_1[i].m then
23:                    T_1[i].m ← T_1[i].m ∧ (¬T_2[j].m)
24:                    remove flow entry j from T_2
25:                else if T_1[i].m ∧ T_2[j].m ≠ ∅ then
26:                    T_1[i].m ← T_1[i].m ∧ (¬T_2[j].m)
27:                    T_2[j].m ← T_2[j].m ∧ (¬T_1[i].m)
28:                end if
29:            end if
30:            if j = T_2.cnt − 1 then
31:                return False
32:            end if
33:        end for
34:    end for
35: end procedure
```

3.3.2 ACA Conversion

Action oriented forwarding set is a collection of items with a key and a value attached to each item, i.e., a dictionary. In this structure, *key* is "action" while *value* represents the conditions that trigger this action. The format and content of the aforementioned actions have been specified in Section 5.8 of OpenFlow 1.5 specification [90]. If two forwarding sets can be transformed into two dictionaries in which all keys and the associated values are identical, they are equivalent.

If multiple entries share the same action, all this action's match fields will be merged with the logical *AND(∧)* operation. However, the match fields in one table are not always independent due to the attribute: "priority". A packet matches against all forwarding tables in the sequence of decreasing priority, which implies that the condition to trigger an action is that this packet will NOT match any item with higher priority. When a forwarding set is reorganised into a dictionary-based structure, the "priority" must be removed. Fig. 3.6 illustrates the deprioritisation process (removing the priority attribute while preserving the same forwarding functionality) with a five-entries example. In Table 1 (the original table), each item is associated with a priority which determines the matching sequence of an incoming packet. After adding an "unmatch" attribute, the dependence among the items have been removed, which means their positions in the table are exchangeable and forwarding functionality remains.

Algorithm 3 depicts the process of removing priority and merging match fields while preserving forwarding functionality. In this algorithm, the process of deprioritisation has been transformed to the *AND(∧)* operation between each item's match fields and the negation of all the match fields with higher priorities.

The core idea of converting multiple tables into a dictionary is achieved by a mapping from the linked table structure to a trie in which each edge represents a match field (Fig. 3.7). Each leaf node represents an action set which can only be reached by the packets matching all the fields along the

Table 1

Match	Action	Priority
M1_1, M1_2	A1	100
M2_1	A2	90
M3_1, M3_2, M3_3	A1	80
M4_1, M4_2	A4	70
*	A2	0

Table 2

Match	Unmatch	Action
M1_1, M1_2	NULL	A1
M2_1	M1_1, M1_2	A2
M3_1, M3_2, M3_3	M1_1, M1_2, M2_1	A1
M4_1, M4_2	M1_1, M1_2, M2_1, M3_1, M3_2, M_3_3	A4
*	M1_1, M1_2, M2_1, M3_1, M3_2, M_3_3, M4_1, M4_2	A2

Figure 3.6: Single table preprocessing: deprioritisation

path from root to that leaf node. In this example, there are four different actions: $\{A_{1\_1}, A_{1\_2}, A_{2\_1}, A_{2\_2}\}$, they are the "keys" in the converted forwarding sets while their associated match fields are determined by the fields in each respective path, which are the composite attributes: $\{M_{0\_1}, M_{1\_1}\}$, $\{M_{0\_1}, M_{1\_2}\}$, $\{M_{0\_2}, M_{2\_1}\}$ and $\{M_{0\_2}, M_{2\_2}\}$.

In a real-world scenario, the structure of a multi-table pipeline is more complicated than the example in Fig. 3.7. Usually a table is composed of multiple entries which contain more than one "goto-table" ids. Here the id is the identification of a forwarding table, for example, the table ids in 3.7 are 0, 1 and 2. Correspondingly, it is very common that there exist more than one table whose traffic is directed to a same table, which means multiple tables share the same "goto-table" id. The table represented by the id should be cloned for all the tables if they contain "goto-table" action with the same id. The process is illustrated in Algorithm 4. The algorithm guarantees that the same "goto-table" id in different tables will be directed to different table. In the converted trie, the number of nodes (denoted

Algorithm 3 Single table preprocessing: deprioritisation and merging

1: $e \leftarrow$ input: forwarding table entries in table T
2: $d \leftarrow$ output: equivalent forwarding dictionary set
3: $.cnt \leftarrow$ count of an array
4: $.m$, $.um$, $.act \leftarrow$ match fields, unmatch fields, actions
5: **procedure** $Single\_table\_Preprocessing(T)$
6: $\quad e_0.um \leftarrow \emptyset$
7: $\quad d.key[e_0.act] \leftarrow e_0.m$
8: $\quad$ **for** $i = 1$ *to* $T.cnt - 1$ **do**
9: $\quad\quad e_i.um \leftarrow e_{i-1}.m \vee e_{i-1}.um$
10: $\quad\quad d.key[e_i.act] \leftarrow e_i.m \wedge (\neg e_i.um)$
11: $\quad$ **end for**
12: **end procedure**

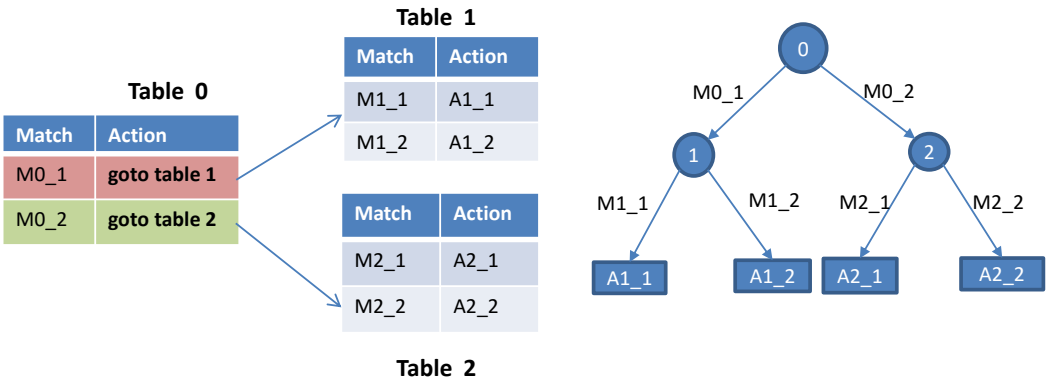


Figure 3.7: Multi-table trie conversion

by *total* in Algorithm 4) is the same or greater than the number of tables due to the clone processing (Line 11 in Algorithm 4). Once a clone happens, the original "goto-table" id must be replaced by the new generated table id (Line 12 in Algorithm 4). An example of this algorithm will be demonstrated in Fig. 3.26 (Function test scenario 4 in section 3.4).

The process to compute a dictionary is detailed in Algorithm 5 which uses a typical breadth-first search strategy. All entries in one table are iterated and the "goto-table-id" in these entries will be replaced by the combinations of this entry and all entries in its associated table. Finally all

Algorithm 4 Multi-table trie conversion

```
1: T ← input: An array of all tables in a switch
2: T′ ← output: An array of tables converted from T
3: D ← A dictionary [key, value] - [original table id, cloned table id]
4: procedure Multi_table_conversion(T, T′)
5:     T′[0] ← T[0]
6:     for i = 0; i ≤ total; i++ do
7:         S ← a set of all ids in table T′[i]'s "goto table" actions
8:         S′ ← ascending sort for all table ids in S
9:         for j = 1; j ≤ S′.count; j++ do
10:            build a dictionary with D[S′(j)] = total + j
11:            clone table T′[D[S′(j)]] ← T[S′(j)]
12:            replace goto table id S′(j) with D[S′(j)] in T′[D[S′(j)]]
13:            total++
14:        end for
15:    end for
16: end procedure
```

the "goto-table-id" will be removed, which means the remaining actions become the "key" in the final dictionary and the match fields in these new generated entries become their corresponding "value".

3.3.3 MFA vs ACA

The concept of EFS means that if the same packet traverses through two EFSs, the same forwarding behaviour they will have, which can be explained from two perspectives: i) the same match fields must share the same action; and ii) the same actions must be associated with the same match fields. Hence, the evaluation of two EFSs can be achieved either by the comparison of actions for the same match field or by the comparison of match fields for the same action.

Both MFA and ACA are built on the comparison of match fields. The difference between them lies in all the match fields of the same action in ACA being combined together while MFA intending to preserve original

Algorithm 5 Multi-table trie traversal

```
1: T ← input: An array of all tables in a switch
2: D ← output: EFS Dictionary [key, value] - [action, match]
3: .cnt ← Properties: Count of an array
4: .m, .act ← Match fields and actions of an entry
5: procedure Multi_table_traversal(T, D)
6:     for i to T.cnt − 1 do
7:         for ek ∈ T[i] do
8:             if goto_table_id ∉ ek.act then
9:                 D[ek.act] ← ek.m ∪ D[ek.act]
10:            else
11:                j ← ek.goto_table_id
12:                for el ∈ T[j] do
13:                    e.m ← el.m ∧ ek.m
14:                    e.act ← el.act ∪ ek.act
15:                    remove goto in ek from el.act
16:                    insert e into T[j]
17:                    remove el from T[j]
18:                end for
19:            end if
20:        end for
21:    end for
22: end procedure
```

match fields.

MFA enumerates all the combinations among the match fields in different tables while ACA constructs the union set for the same actions. Take the multi-table forwarding set in Fig. 3.4 as an example, there is a total number of 14 entries in all three tables. However, as shown in Fig. 3.8, the combinations of all the potential match fields based on their association relationship among these three tables will result in 24 entries. However, as shown in Fig. 3.9, the total number of entries in ACA is reduced to only eight due to the limited actions.

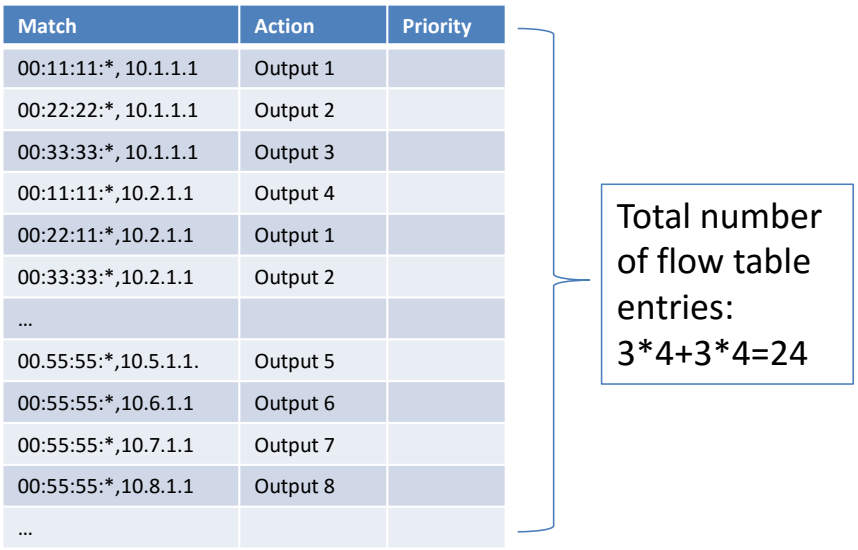


Figure 3.8: Match-fields oriented conversion approach (MFA) yielding a "tall" table.

3.4 Functional Test and Performance Evaluation

3.4.1 Functional Test Design

The function test adopts black box model in which the switch is a black box while the incoming packets and outgoing packets are this box's input and output, respectively (Fig. 3.10). In each test, the same incoming packets will be sent to the switch in which the forwarding set will be presented in the different form (Original MFT, Equivalent MFA and ACA).

Open vSwitch has been adopted as a networking operating system by many platforms (for example, OpenStack) and physical switches (for example, Pica8). To balance the flexibility and applicability, the latest Open vSwitch 2.6.1 [93] is chosen as the test platform. It maintains the exact same design and implementation as the one on the physical switch but offers stronger multi-table capability. Scapy [94] is used for packet generation due to its flexible customization ability. More details about the test bed setup are listed in Appendix D.

Action	Match
Output 1	(00:11:11:*, 10.1.1.1) V (00:22:22:*, 10.1.1.1) V (00:33:33:*, 10.1.1.1)
Output 2	(00:11:11:*, 10.2.1.1) V (00:22:22:*, 10.2.1.1) V (00:33:33:*, 10.2.1.1)
Output 3	(00:11:11:*, 10.3.1.1) V (00:22:22:*, 10.3.1.1) V (00:33:33:*, 10.3.1.1)
Output 4	(00:11:11:*, 10.4.1.1) V (00:22:22:*, 10.4.1.1) V (00:33:33:*, 10.4.1.1)
Output 5	(00:44:44:*, 10.5.1.1) V (00:55:55:*, 10.5.1.1) V (00:66:66:*, 10.5.1.1)
Output 6	(00:44:44:*, 10.6.1.1) V (00:55:55:*, 10.6.1.1) V (00:66:66:*, 10.6.1.1)
Output 7	(00:44:44:*, 10.7.1.1) V (00:55:55:*, 10.7.1.1) V (00:66:66:*, 10.7.1.1)
Output 8	(00:44:44:*, 10.8.1.1) V (00:55:55:*, 10.8.1.1) V (00:66:66:*, 10.8.1.1)

Total number of flow table entries: 8

Figure 3.9: Action oriented conversion approach (ACA) yielding a "fat" table.

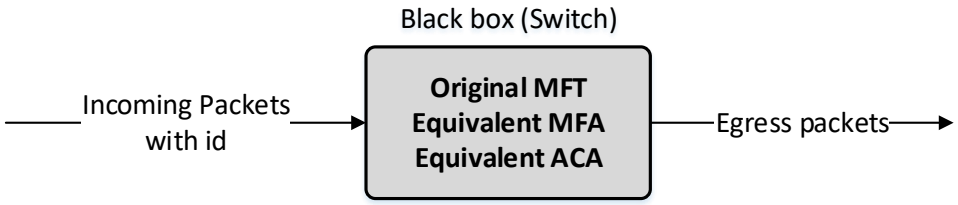


Figure 3.10: Test environment design: functional test principle

3.4.1.1 ACA Forwarding Table Deployment

In the conversion process from an arbitrary table to a MFA or ACA table, the original MFT and its equivalent MFA are still compatible with the existing switch (for example, Open vSwitch), which means these tables are able to be installed into a switch without any transformation. Compared to MFA, the match fields in ACA can also be transformed into conjunctive normal form (CNF). However, each clause in ACA's CNF is not a simple match field, it becomes a disjunctive normal form (DNF) of the simple match fields and their negatives. Thus they are not able to be written into the OpenFlow table in real switch directly.

Algorithm 6 illustrates the process to convert a single action's associated match fields into a switch compatible table. All non-negative fields (a field without unary operation "¬") in a clause will be installed into a single table in sequence (Algorithm 6 Lines 13-15). Every negative field (a field

with an unary operation "¬") occupies a separate table in which an entry with the original non-negative match field to pass packets to next table (or drop if it is already the last table) and a default entry to pass packets to next clause (or execute the action if it is the last table) are installed. These two entries are combined to represent a negative field (Algorithm 6 Lines 18-26). Thus for each clause, the incoming packets will be proceed by the non-negative table firstly and all matched packets will be sent to the first table of next clause. For the unmatched packets, they will be sent to next table(s) to match against the negative fields. With this design, the "OR" operation between two fields inside a clause is represented by the action "goto next table" between any two adjacent tables. The whole process is demonstrated in Fig. 3.18.

Algorithm 6 illustrates the process to verify the forwarding behaviour of a single action in an ACA table. However, the table(s) occupied by an action cannot be shared by other actions. Otherwise, the later installed action's associated match fields with lower priority in a table might not get the chance to match against the incoming packets. Thus, it is not able to deploy an entire ACA table into the existing Open vSwitch, only one action's match fields can be installed in each test. Thus an ACA table's verification will involve multiple rounds of single action based test.

The functional test is to verify if a given MFT table forwards all incoming packets to the same destinations as its converted MFA and AFA table using the approaches in Section 3.3, i.e., to verify whether these three tables are equivalent.

In the black box of Fig. 3.10, three type of forwarding tables will be verified. Among them, the original table(s) and its equivalent MFA are still compatible with Open vSwitch and able to be installed without any transformation. For each action in ACA, the associated match fields can be written into Open vSwitch by Algorithm 6.

Algorithm 6 ACA associated Boolean function deployment

$S \leftarrow$ input: single ACA associated match fields in the form of CNF
C.negCount $\leftarrow$ input: (the count of negative fields in each clause C) + 1
$ct \leftarrow$ input: the index of the table in processing
procedure $Aca\_deployment$
    $ct \leftarrow 0$
    for each clause C in S **do**
        $rt \leftarrow$ *C.negCount*
        if C is the last clause in S **then**
            act $\leftarrow$ *S.action*
        else
            act $\leftarrow$ "goto $(ct + rt)^{th}$ table"
        end if
        for each non-negative field m in C **do**
            install flow {match $= m$, action $= act$} in table ct
        end for
        install flow {match $= *$, action = "goto $(ct + 1)^{st}$ table"} in ct
        $ct \leftarrow ct + 1$
        for each negative field $\neg m$ in C **do**
            if $ct \neq rt - 1$ **then**
                install flow {match $= m$, action = "goto $(ct + 1)^{st}$ table"}
            else
                install flow {match $= m$, action = DROP}
            end if
            install flow {match $= *$, action $= act$} in table ct
            $ct \leftarrow ct + 1$
        end for
    end for
end procedure

3.4.1.2 Functional Test Scenarios

In this section, three typical scenarios in traditional networking are verified: i) Forwarding information base (FIB) for OSI layer 2 switching; ii) simple IP table for layer 3 routing; iii) MPLS to VLAN translation. To cover the various types of forwarding set, a comprehensive multiple inward outward pipeline is also presented in the end of this subsection.

Scenario 1: FIB

A forwarding information base (FIB), also known as a forwarding table or MAC table, is most commonly used in OSI layer 2 switching to find the proper interface to which the input interface should forward a packet. It is a dynamic table that maps MAC addresses to ports based on their VLAN groups. In the case of Fig. 3.11, the packets with VLAN 1001-1002 and destination MAC (DMAC) ending with 11 and 22 will be forwarded to port 1 or 2 while the packets belonging to VLAN 1003-1004 will be forwarded to port 3 or 4 based on their respective DMAC.

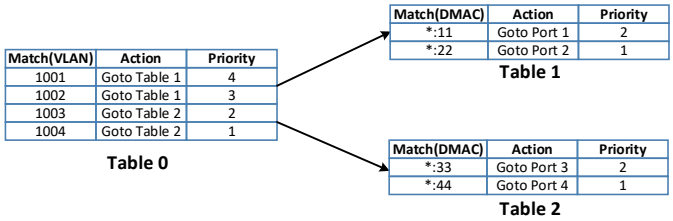


Figure 3.11: Functional test scenario 1 - MAC forwarding

The equivalent MFA and ACA of the test case in Fig. 3.11 has been demonstrated in Fig. 3.12 and Fig. 3.13, respectively. During the verification, all the converted FTEs in MFA are still able to be written in Open vSwitch but the FTEs in ACA cannot be written directly to Open vSwitch because they contain the "OR($\vee$)" operations. With the help of Algorithm 6, after converting "OR($\vee$)" to multiple entries, every action can be verified separately by deploying them to Open vSwitch. The detailed process will be exemplified in the next test scenario.

Match(VLAN ∧ DMAC)	Action	Priority
1001 ∧ *:11	Output port 1	8
1001 ∧ *:22	Output port 2	7
1002 ∧ *:11	Output port 1	6
1002 ∧ *:22	Output port 2	5
1003 ∧ *:33	Output port 3	4
1003 ∧ *:44	Output port 4	3
1004 ∧ *:33	Output port 3	2
1004 ∧ *:44	Output port 4	1

Figure 3.12: Functional test scenario 1 - MFA of the MAC forwarding table in Fig. 3.11

Action	Match(VLAN ∧ DMAC)
Output port 1	(1001 ∨ 1002) ∧ (*:11)
Output port 2	(1001 ∨ 1002) ∧ (*:22)
Output port 3	(1003 ∨ 1004) ∧ (*:33)
Output port 4	(1003 ∨ 1004) ∧ (*:44)

Figure 3.13: Functional test scenario 1 - ACA of the MAC forwarding table in Fig. 3.11

During the test, various type of traffic with the specific VLAN and MAC in the range of match fields in Fig. 3.11 have been fed. As illustrated in Fig. 3.14, in each test, the same packets are replayed for MFT, MFA and ACA to observe their respective actions. In the rest of this subsection, MFT also stands for the original forwarding table which unusually presents in the form of multiple flow table.

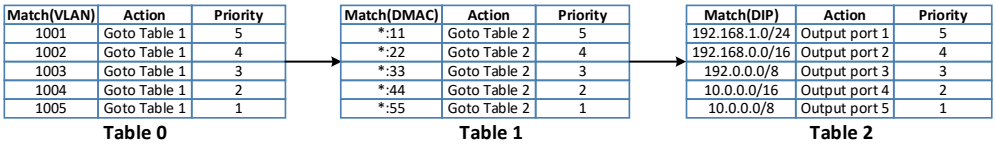
Test Case		MFT	MFA	ACA	Expected Output
VLAN	MAC				
1001	00:0a:95:9d:68:11	✓	✓	✓	Output port 1
1001	00:0a:95:9d:68:22	✓	✓	✓	Output port 2
1002	00:0a:95:9d:68:11	✓	✓	✓	Output port 1
1002	00:0a:95:9d:68:22	✓	✓	✓	Output port 2
1003	00:0a:95:9d:68:33	✓	✓	✓	Output port 3
1003	00:0a:95:9d:68:44	✓	✓	✓	Output port 4
1004	00:0a:95:9d:68:33	✓	✓	✓	Output port 3
1004	00:0a:95:9d:68:44	✓	✓	✓	Output port 4

Figure 3.14: Functional test scenario 1 - summary of the MAC forwarding table in Fig. 3.11

Scenario 2: IP routing table

An IP routing table is used to determine where data packets travelling over an Internet Protocol (IP) network will be directed. The packets matching a certain range of VLAN and destination MAC address will be sent to routing table for IP lookup to determine its output port. In traditional networking, a MAC address usually points to a switch or router rather than any hosts in the same subnet to indicate the packets should be

further forwarded based on its destination IP (DIP) address. Hence, the actual match fields for an IP routing table include VLAN, DMAC and DIP, which are represented in the three separate tables in Fig. 3.15.



Match(VLAN)	Action	Priority
1001	Goto Table 1	5
1002	Goto Table 1	4
1003	Goto Table 1	3
1004	Goto Table 1	2
1005	Goto Table 1	1

Table 0

Match(DMAC)	Action	Priority
*:11	Goto Table 2	5
*:22	Goto Table 2	4
*:33	Goto Table 2	3
*:44	Goto Table 2	2
*:55	Goto Table 2	1

Table 1

Match(DIP)	Action	Priority
192.168.1.0/24	Output port 1	5
192.168.0.0/16	Output port 2	4
192.0.0.0/8	Output port 3	3
10.0.0.0/16	Output port 4	2
10.0.0.0/8	Output port 5	1

Table 2

Figure 3.15: Functional test scenario 2 - IP routing table

For the given IP routing example in Fig. 3.15, its equivalent MFA and ACA are illustrated in Fig. 3.16 and Fig. 3.17, respectively. The match fields in MFA are in conjunctive normal form (CNF) in which every clause is a single match field. Thus it can be easily verified by Open vSwtich. However, the ACA table cannot be written to Open vSwitch directly because it contains the "OR($\vee$)" and "NOT($\neg$)" operations.

Match(VLAN $\wedge$ DMAC $\wedge$ DIP)	Action	Priority
1001 $\wedge$ *:11 $\wedge$ (192.168.1.0/24)	Output port 1	125
1001 $\wedge$ *:11 $\wedge$ (192.168.0.0/16)	Output port 2	124
.....	.....	.....
1001 $\wedge$ *:11 $\wedge$ (10.0.0.0/8)	Output port 5	121
1001 $\wedge$ *:22 $\wedge$ (192.168.1.0/24)	Output port 1	120
.....	.....	.....
1001 $\wedge$ *:22 $\wedge$ (10.0.0.0/8)	Output port 5	116
.....	.....	.....
1005 $\wedge$ *:55 $\wedge$ (10.0.0.0/8)	Output port 5	1

Figure 3.16: Functional test scenario 2 - MFA of the IP routing table in Fig. 3.15

Action	Match(VLAN $\wedge$ DMAC $\wedge$ DIP)
Output port 1	(1001 $\vee$ 1002 $\vee$ 1003 $\vee$ 1004 $\vee$ 1005) $\wedge$ (*:11 $\vee$ *:22 $\vee$ *:33 $\vee$ *:44 $\vee$ *:55) $\wedge$ (192.168.1.0/24)
Output port 2	(1001 $\vee$ 1002 $\vee$ 1003 $\vee$ 1004 $\vee$ 1005) $\wedge$ (*:11 $\vee$ *:22 $\vee$ *:33 $\vee$ *:44 $\vee$ *:55) $\wedge$ ((192.168.0.0/16) $\wedge$ ($\neg$(192.168.1.0/24)))
Output port 3	(1001 $\vee$ 1002 $\vee$ 1003 $\vee$ 1004 $\vee$ 1005) $\wedge$ (*:11 $\vee$ *:22 $\vee$ *:33 $\vee$ *:44 $\vee$ *:55) $\wedge$ ((192.0.0.0/8) $\wedge$ ($\neg$(192.168.0.0/16)))
Output port 4	(1001 $\vee$ 1002 $\vee$ 1003 $\vee$ 1004 $\vee$ 1005) $\wedge$ (*:11 $\vee$ *:22 $\vee$ *:33 $\vee$ *:44 $\vee$ *:55) $\wedge$ (10.0.0.0/16)
Output port 5	(1001 $\vee$ 1002 $\vee$ 1003 $\vee$ 1004 $\vee$ 1005) $\wedge$ (*:11 $\vee$ *:22 $\vee$ *:33 $\vee$ *:44 $\vee$ *:55) $\wedge$ ((10.0.0.0/8) $\wedge$ ($\neg$(10.0.0.0/16)))

Figure 3.17: Functional test scenario 2 - ACA of the IP routing table in Fig. 3.15

Fig. 3.18 illustrates the deployment of an ACA table on Open vSwitch for action "output port 5" in Fig. 3.17 by applying Algorithm 6. Here four tables are used to represent the equivalent match fields of a single action in ACA. Once the same packets are replayed in the switch, all packets egressing from port 5 will be identified. This process is repeated for other

output ports until all ACA's actions have been tested.

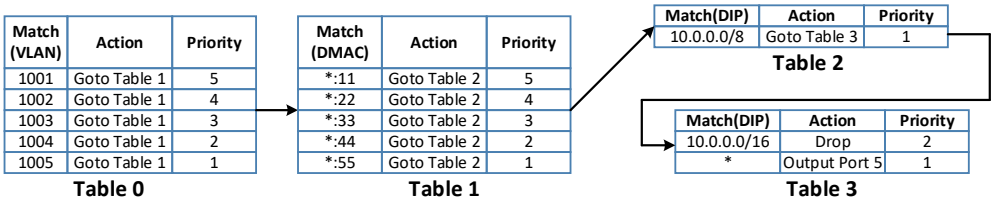


Figure 3.18: Functional test scenario 2 - the deployment of the ACA with action "Output port 5" in Fig. 3.15

During the functional test, the various combinations of VLAN, DMAC and DIP in the range specified in Fig. 3.15 have been tested. As shown in Fig. 3.19, the packets matching multiple entries in MFT are forwarded based on their original priorities. For example, the packets with destination IP 192.168.1.1 (row 3 in Fig. 3.19) which matches the first two entries in Table 2 (Fig. 3.15) are correctly sent to output port 1 in all three tables.

Test Case			MFT	MFA	ACA	Expected Output
VLAN	DMAC	DIP				
1001	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
1001	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
1002	00:0a:95:9d:68:33	192.1.1.1	✓	✓	✓	Output port 3
1002	00:0a:95:9d:68:44	10.0.1.1	✓	✓	✓	Output port 4
1003	00:0a:95:9d:68:55	10.1.1.1	✓	✓	✓	Output port 5
1003	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
1004	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
1004	00:0a:95:9d:68:33	192.1.1.1	✓	✓	✓	Output port 3
1005	00:0a:95:9d:68:44	10.0.1.1	✓	✓	✓	Output port 4
1005	00:0a:95:9d:68:55	10.1.1.1	✓	✓	✓	Output port 5

Figure 3.19: Functional test scenario 2 - summary of the IP routing table in Fig. 3.15

Scenario 3: MPLS to VLAN translation

Another typical networking scenario is the Multiprotocol Label Switching (MPLS) label push, swap and pop. When a packet passes through the gateway from wide area network (WAN) to local area network (LAN), the MPLS label it carries will be popped and replaced by the local VLAN tag.

The matching condition of MPLS to VLAN translation can be represented by the tables in Fig. 3.20 in which the ingress port and DMAC are

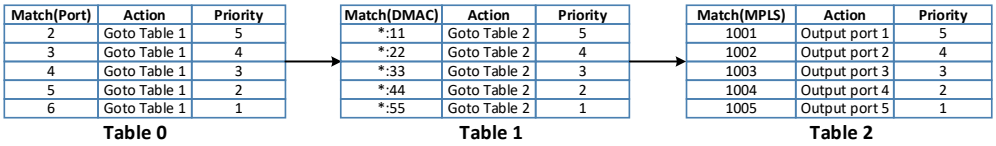


Figure 3.20: Functional test scenario 3 - MPLS to VLAN translation

added as match fields because they are often used to identify whether a packet should be further processed with MPLS operations. In a real MPLS translation case, the actions include MPLS pop, VLAN tagging, source and destination MAC replacement, here only output port is specified because MFA and ACA focus on the conversion of match fields.

Match (Port ∧ DMAC ∧ MPLS)	Action	Priority
2 ∧ *:11 ∧ 1001	Output port 1	125
2 ∧ *:11 ∧ 1002	Output port 2	124
.....	.....	.....
2 ∧ *:11 ∧ 1005	Output port 5	121
2 ∧ *:22 ∧ 1001	Output port 1	120
.....	.....	.....
2 ∧ *:22 ∧ 1005)	Output port 5	116
.....	.....	.....
6 ∧ *:55 ∧ 1005	Output port 5	1

Figure 3.21: Functional test scenario 3 - MFA of the MPLS to VLAN translation in Fig. 3.20

Action	Match(Port ∧ DMAC ∧ MPLS)
Output port 1	(2 ∨ 3 ∨ 4 ∨ 5 ∨ 6) ∧ (*:11 ∨ *:22 ∨ *:33 ∨ *:44 ∨ *:55) ∧ 1001
Output port 2	(2 ∨ 3 ∨ 4 ∨ 5 ∨ 6) ∧ (*:11 ∨ *:22 ∨ *:33 ∨ *:44 ∨ *:55) ∧ 1002
Output port 3	(2 ∨ 3 ∨ 4 ∨ 5 ∨ 6) ∧ (*:11 ∨ *:22 ∨ *:33 ∨ *:44 ∨ *:55) ∧ 1003
Output port 4	(2 ∨ 3 ∨ 4 ∨ 5 ∨ 6) ∧ (*:11 ∨ *:22 ∨ *:33 ∨ *:44 ∨ *:55) ∧ 1004
Output port 5	(2 ∨ 3 ∨ 4 ∨ 5 ∨ 6) ∧ (*:11 ∨ *:22 ∨ *:33 ∨ *:44 ∨ *:55) ∧ 1005

Figure 3.22: Functional test scenario 3 - ACA of the MPLS to VLAN translation in Fig. 3.20

Since the original MFT of the MPLS translation is very similar to the IP routing table in second scenario, they have similar MFA and ACA (Fig. 3.21 and Fig. 3.22). A difference is that MPLS does not support wildcard, thus the deprioritisation can be avoided, which makes the ACA deployment easier because every single action only requires one table. (In Fig. 3.18, the DIP for action output port 5 takes two tables due to the existence of IP overlap.) The test cases to verify this scenario are also listed in Fig. 3.23.

Scenario 4: A multiple-inward multiple-outward case

Test Case			MFT	MFA	ACA	Expected Output
Port	DMAC	MPLS				
2	00:0a:95:9d:68:11	1001	✓	✓	✓	Output port 1
2	00:0a:95:9d:68:22	1002	✓	✓	✓	Output port 2
3	00:0a:95:9d:68:33	1003	✓	✓	✓	Output port 3
3	00:0a:95:9d:68:44	1004	✓	✓	✓	Output port 4
4	00:0a:95:9d:68:55	1005	✓	✓	✓	Output port 5
4	00:0a:95:9d:68:11	1001	✓	✓	✓	Output port 1
5	00:0a:95:9d:68:22	1002	✓	✓	✓	Output port 2
5	00:0a:95:9d:68:33	1003	✓	✓	✓	Output port 3
6	00:0a:95:9d:68:44	1004	✓	✓	✓	Output port 4
6	00:0a:95:9d:68:55	1005	✓	✓	✓	Output port 5

Figure 3.23: Functional test scenario 3 - summary of the IP routing table in Fig. 3.20

The last test scenario depicts the conversion of a multiple inward outward MFT. Fig. 3.24 demonstrates a case which contains tables (Table 0 and Table 1) who have two outward links and tables (Table 2 and Table 3) who have two inward links.

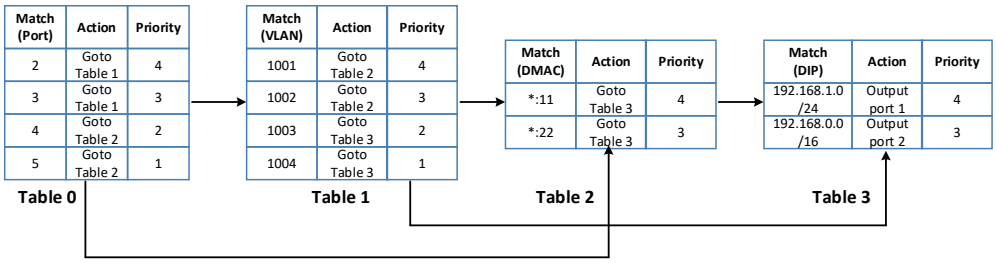


Figure 3.24: Functional test scenario 4 - a multiple-inward multiple-outward case

Figure 3.25 presents the equivalent single table which is converted from the multi-table example in Fig. 3.24. All the flows in the converted MFA table are the compositions of the original match fields and they are still able to write into Open vSwitch, thus the same method for the other scenarios can be used to verify the equivalence of MFA in Fig. 3.25.

In the processing of ACA computation, the conversion from the original pipeline to the trie structure is different from the previous simple cases due to the existence of the multi-inward links. To make Algorithm 5 applicable to this case, all the inward links need to be removed by the

Match				Action	Priority
Port	VLAN	DMAC	DIP		
2	1001	*:11	192.168.1.0/24	Output port 1	32
2	1001	*:11	192.168.0.0/16	Output port 2	31
2	1001	*:22	192.168.1.0/24	Output port 1	30
2	1001	*:22	192.168.0.0/16	Output port 2	29
2	1002	*:11	192.168.1.0/24	Output port 1	28
2	1002	*:11	192.168.0.0/16	Output port 2	27
2	1002	*:22	192.168.1.0/24	Output port 1	26
2	1002	*:22	192.168.0.0/16	Output port 2	25
2	1003	*	192.168.1.0/24	Output port 1	24
2	1003	*	192.168.0.0/16	Output port 2	23
2	1004	*	192.168.1.0/24	Output port 1	22
2	1004	*	192.168.0.0/16	Output port 2	21
2	1001	*:11	192.168.1.0/24	Output port 1	20
2	1001	*:11	192.168.0.0/16	Output port 2	19
3	1001	*:22	192.168.1.0/24	Output port 1	18
3	1001	*:22	192.168.0.0/16	Output port 2	17
3	1002	*:11	192.168.1.0/24	Output port 1	16
3	1002	*:11	192.168.0.0/16	Output port 2	15
3	1002	*:22	192.168.1.0/24	Output port 1	14
3	1002	*:22	192.168.0.0/16	Output port 2	13
3	1003	*	192.168.1.0/24	Output port 1	12
3	1003	*	192.168.0.0/16	Output port 2	11
3	1004	*	192.168.1.0/24	Output port 1	10
3	1004	*	192.168.0.0/16	Output port 2	9
4	*	*:11	192.168.1.0/24	Output port 1	8
4	*	*:11	192.168.0.0/16	Output port 2	7
4	*	*:22	192.168.1.0/24	Output port 1	6
4	*	*:22	192.168.0.0/16	Output port 2	5
5	*	*:11	192.168.1.0/24	Output port 1	4
5	*	*:11	192.168.0.0/16	Output port 2	3
5	*	*:22	192.168.1.0/24	Output port 1	2
5	*	*:22	192.168.0.0/16	Output port 2	1

Figure 3.25: Functional test scenario 4 - MFA of the multiple-inward multiple-outward case in Fig. 3.24

clone process implemented in Algorithm 4. The left trie in Fig. 3.26 depicts the actual table jumping relationship in which a same table might be traversed for more than one time. The right trie converts the original four-nodes trie into a six-nodes trie by cloning and reindexing all nodes in sequence. After this transformation, the equivalent ACA can be acquired following the process in Algorithm 5.

The final result shown in Fig. 3.27 is a composition of all match fields for the same action (Output port 1 and 2) in Fig. 3.24. Here the match fields are separated according to their respective traversing path to bet-

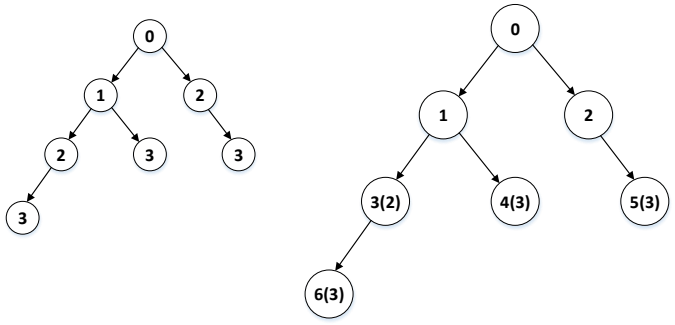


Figure 3.26: Functional test scenario 4 - trie conversion for the multiple-inward multiple-outward case in Fig. 3.24

ter demonstrate the packet matching sequence in the original forwarding pipeline. The correctness of the ACA conversion can also be proven with the same deployment strategy illustrated in Algorithm 6.

Action	**Match**				**Traverse path**
	Port	VLAN	DMAC	DIP	
Output port 1	2 ∨ 3	1003 ∨ 1004		192.168.1.0/24	0-1-4(3)
Output port 2	2 ∨ 3	1003 ∨ 1004		¬(192.168.1.0/24) ∧ (192.168.0.0/16)	
Output port 1	4 ∨ 5		*:11 ∨ *:22	192.168.1.0/24	0-2-5(3)
Output port 2	4 ∨ 5		*:11 ∨ *:22	¬(192.168.1.0/24) ∧ (192.168.0.0/16)	
Output port 1	2 ∨ 3	1001 ∨ 1002	*:11 ∨ *:22	192.168.1.0/24	0-1-3(2)-6(3)
Output port 2	2 ∨ 3	1001 ∨ 1002	*:11 ∨ *:22	¬(192.168.1.0/24) ∧ (192.168.0.0/16)	

Figure 3.27: Functional test scenario 4 - ACA result of the multiple-inward multiple-outward case in Fig. 3.24

Figure 3.28 summarises all the test cases in this scenario. All the ports, VLAN, DMAC and DIP in the original MFT are enumerated to guarantees its equivalence to MFA and ACA.

This multi-inward multi-outward scenario can be interpreted as the various conditions for a packet to be proceeded by an IP routing table, which are the combination of: i) ingress port and VLAN; ii) DMAC and DIP; iii) Port, VLAN, and DMAC. More importantly, this case represents a generic MFT structure, which proves the feasibility of MFA and ACA.

Test Case				MFT	MFA	ACA	Expected Output
Port	VLAN	DMAC	DIP				
2	1001	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
2	1001	00:0a:95:9d:68:11	192.168.0.1	✓	✓	✓	Output port 2
2	1001	00:0a:95:9d:68:22	192.168.1.1	✓	✓	✓	Output port 1
2	1001	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
2	1002	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
2	1002	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
2	1002	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
2	1002	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
2	1003	00:0a:95:9d:68:EE	192.168.1.1	✓	✓	✓	Output port 1
2	1003	00:0a:95:9d:68:EE	192.168.0.1	✓	✓	✓	Output port 2
2	1004	00:0a:95:9d:68:EE	192.168.1.1	✓	✓	✓	Output port 1
2	1004	00:0a:95:9d:68:EE	192.168.0.1	✓	✓	✓	Output port 2
3	1001	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
3	1001	00:0a:95:9d:68:11	192.168.0.1	✓	✓	✓	Output port 2
3	1001	00:0a:95:9d:68:22	192.168.1.1	✓	✓	✓	Output port 1
3	1001	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
3	1002	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
3	1002	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
3	1002	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
3	1002	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
3	1003	00:0a:95:9d:68:aa	192.168.1.1	✓	✓	✓	Output port 1
3	1003	00:0a:95:9d:68:bb	192.168.0.1	✓	✓	✓	Output port 2
3	1004	00:0a:95:9d:68:cc	192.168.1.1	✓	✓	✓	Output port 1
3	1004	00:0a:95:9d:68:dd	192.168.0.1	✓	✓	✓	Output port 2
4	2001	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
4	2002	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
4	2003	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
4	2004	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
5	2005	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
5	2006	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2
5	2007	00:0a:95:9d:68:11	192.168.1.1	✓	✓	✓	Output port 1
5	2008	00:0a:95:9d:68:22	192.168.0.1	✓	✓	✓	Output port 2

Figure 3.28: Functional test scenario 4 - summary of the multiple-inward multiple-outward case in Fig. 3.24

3.4.2 Performance Evaluation

3.4.2.1 Complexity Analysis

The EFS evaluation consists of two stages: i) the conversion from any arbitrary types of forwarding tables into either MFA or ACA table; ii) the comparison between two converted MFA or ACA tables. Both stages are involved with large amount of space-hungry and time-consuming Boolean operations. The details will be investigated in the following complexity analysis, moreover, the potential methods to reduce the computation time

and the respective applicable scenarios for MFA and ACA will also be discussed.

Table 3.1 compares the number of FTEs in the original MFTs and their corresponding converted MFA and ACA tables for the four scenarios in functional tests. The number of MFA (denoted as N_M) is determined by the number of tables and the number of FTE in each table while the number of ACA (denoted as N_A) is only associated with the final action(s). Supposing there is a MFT with t forwarding tables, f FTEs and a actions, the maximum number of MFA will be $(\frac{f}{t})^t$ while N_A is always the same as a. For a MFT with two linked tables in which one contains 100 MAC address entries and the other contains 100 IP addresses, the total number of entries in the converted single table by MFA will reach $100 \times 100 = 10,000$ which is significantly higher than the original total number of entries (just $100 + 100 = 200$). However, the size of its converted ACA table is determined by the number of actions which is smaller than the original MFT.

Table 3.1: Total number of entries in original multiple flow tables, MFA and ACA for afore-mentioned four function test scenarios

	Scenario	**Match Field**	**Type**	N_O	N_M	N_A
1	FIB	VLAN, DMAC	Multiple outward	8	8	4
2	IP routing table	VLAN, DMAC, DIP	Sequential	15	125	5
3	MPLS to VLAN	Port, DMAC, MPLS	Sequential	15	125	5
4	Comprehensive case	Port, VLAN, DMAC, DIP	Multiple inward, multiple outward	12	32	2

Notes:
N_O - Number of FTEs in original MFT.
N_M - Number of FTEs in equivalent MFA.
N_A - Number of FTEs in equivalent ACA.

MFA introduces not only the greater number of FTE but also the higher complexity. As illustrated in Table 3.2, the conversion complexity of MFT is proportional to $(\frac{f}{t})^t$ because every entry in a table operates with the FTEs in all other tables. The conversion of ACA has two steps: i) the removal of priorities and the merging of the FTEs for the same action; ii) the trie conversion and traversal. The first step occurs in each individual

forwarding table and the complexity is proportional to the number of FTE ($\mathcal{O}(f)$). The second step intends to combine all the action indexed FTEs in different tables whose complexity is determined by the number of tables and their containing actions ($\mathcal{O}(\frac{a}{t})^t$).

The complexity of the second step is determined by the number of FTEs in the converted tables. In the worst case, two equivalent MFTs with different type of structure might yield two completely different MFA tables, therefore, all FTEs in one table must compare with all FTEs in anther MFA, this is why the comparison complexity is $\mathcal{O}((\frac{f}{t})^{2t})$ rather than $\mathcal{O}((\frac{f}{t})^t)$. This will not happen on ACA because all the FTEs have been indexed based on actions.

Besides MFA and ACA, the closest research on equivalent forwarding set is MTO which aims to convert M-stage multiple flow tables into a One-stage flow table [15]. It can be observed that MTO chooses a similar strategy as MFA thus they have the same complexity. However, MTO has limited applicable networking scenarios because it did not notice the effect of priority and the existences of multiple inward and outward structure.

Table 3.2: Complexity comparison of MFA, ACA and MTO

	MFA	**ACA**	**MTO**
Conversion complexity	$\mathcal{O}((\frac{f}{t})^t)$	$\mathcal{O}(f + (\frac{a}{t})^t)$	$\mathcal{O}((\frac{f}{t})^t)$
Comparison complexity	$\mathcal{O}((\frac{f}{t})^{2t})$	$\mathcal{O}(a)$	$\mathcal{O}((\frac{f}{t})^{2t})$

From table 3.1 and 3.2, it can be observed that ACA has lower complexity than MFA, however, it does not imply that MFA always requires less storage space and conversion time. It only indicates that ACA has fewer Boolean operations than MFA but each operation might take longer time because ACA retains all the original match fields for the same action. In the following two subsections, the performances of MFA and ACA are evaluated and the way to reduce the computation time are also investigated.

3.4.2.2 Formula Based Boolean Operation

The storage and computation of Boolean functions are proceeded in the format of either formula or truth-table (Appendix C). Equation C.3 demonstrates that the number of the prime implicants is constrained by variables and conjunctions. Take IPv4 which requires 32 bits (as thus requires 32 variables) as an example, it will have at least $3^{\lfloor 32/3 \rfloor} = 3^{10} = 59,049$ prime implicants if no limitation of conjunctions imposed. In practice, the width of match fields might be far larger than 32 due to the existence of various fields such as MAC address, VLAN ID, MPLS label and their combinations. Usually these values spread in various ranges and they are hard to be merged to eliminate the variables significantly, so the reduction of prime implicants by the elimination of variables is difficult to achieve.

Referring again to Eq. (C.3), another way to limit the number of prime implicants is to reduce the number of conjunctions. In most cases, the number of variables should be in the same order-of-magnitude for MFA and ACA because both of them rely on the same composition of match fields. It is possible for MFA to use fewer variables to represent a match field because it involves far less conjunctions than ACA, however, the number of prime implicants is still considerably large. Suppose there are two forwarding entries with different match fields, for example, one IP address and one MAC address, if they share the same action, they will be merged into one forwarding set ultimately in ACA, which means the variables in ACA will become the sum of IP and MAC address. While in MFA, they should always belong two different forwarding sets and they will be compared in their respective width.

Though there is not much difference on the number of variables between ACA and MFA, their conjunctions vary significantly. In ACA, all match fields with the same action will be merged into one big set which contains all the conjunctions from the original forwarding entries. The situation becomes worse during the process of deprioritisation. In deprioritisation, the compliment of the match fields with higher priority will be introduced,

which is huge compared to the original conjunctions. Take a forwarding table with only two entries as an example, if both of them contain an IPv4 address (IP_1 and IP_2) as match fields while their actions vary, the match fields of the second entry will become $\{IP_1 \wedge (\neg\, IP_2)\}$ after deprioritisation. According to the wildcard-supported "compliment" operations, it will have 32 conjunctions. Thus in the final ACA forwarding set, the number of conjunctions equals to the sum of original conjunctions and their compliments. However, this problem does not exist in MFA which does not involve deprioritisation and merging among multiple forwarding entries.

In summary, ACA defers the normalization and comparison to the final step of the conversion process and then compares the generated large and complexed forwarding sets, the final number of conjunctions are usually a factor ten more than the number of original entries. Because of this, the computation of prime implicants becomes extremely complicated and the number of prime implications turns to extraordinary large. However, ACA is easily implemented in real-world applications with the help of truth table, which makes it more favourable in practice.

3.4.2.3 Truth-table Based Boolean Operation

Another way to compare two Boolean functions is to convert them into truth tables and then compare all the elements one by one. This is achieved by electronic logic gates or software. The comparison of two Boolean values is an exclusive logical OR operation. The performance on two Boolean functions with different widths and lengths is illustrated in Fig. 3.29.

Fig. 3.29 demonstrates the time consumption of bitwise operation between two truth-tables on a Dell OptiPlex 9020 server with Intel i7 Quad-Core CPU and 8GB RAM. The conversion time from Boolean functions to truth-table is excluded. It is observed that the execution time is not sensitive to the width of Boolean values. The comparison time of 128 bits width truth table is only increased around 14.4% compared with the 32 bits table

in the case of 2^{32} elements. However, the time is linearly proportional to the number of elements, it reaches 24 minutes for the comparison of two 2^{32} elements. This result demonstrates that the number of Boolean values is the decisive factor in the comparison of two Boolean functions. Hence, the truth-table based comparison favours ACA than MFA because it has smaller number of FTE.

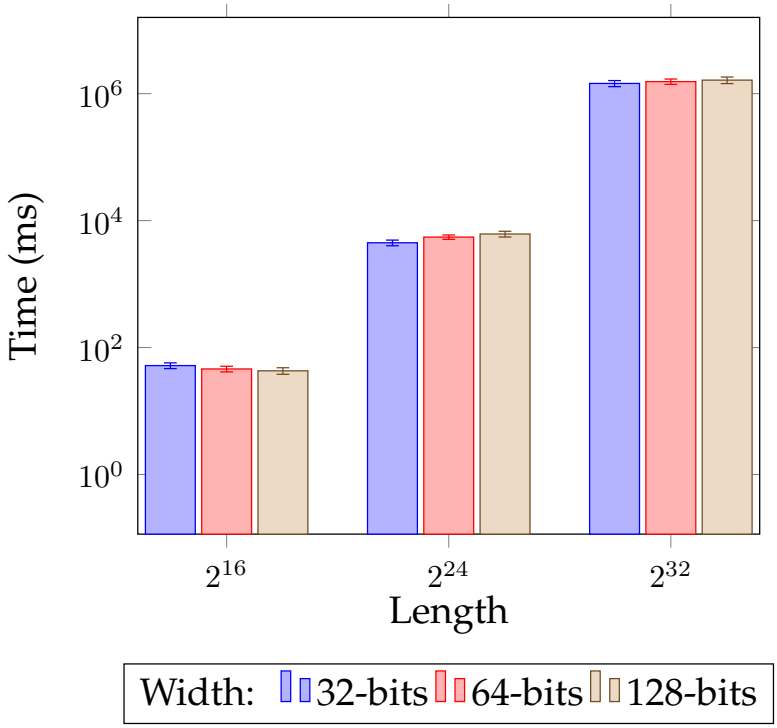


Figure 3.29: Time of Boolean operations with variable lengths and widths

In the conversion between Boolean function and a truth table, the length of a truth table can be further reduced with the introduction of wildcard. In this scenario, each element must strictly represent one prime implicant, otherwise, the same Boolean functions with different form of truth tables might be determined as inequivalent.

Therefore, the performance of formula-based approach primarily relies on the number of prime implicants which are determined by width and length while truth-table based approach values length more than width.

Since the width in a forwarding set is fixed (determined by the match fields in original MFT), reducing the complexity relies on reducing the number of length, i.e., reducing the number of operations. Thus, ACA achieves better performance than MFA because it intends to push merging operations ahead in every stage, for example, in deprioritisation and trie traversal. However, MFA can be easily verified by hardware or Open vSwitch because the match fields of MFA are the simple "AND($\wedge$)" operation of all the original individual match filed, which makes them still compatible with the existing forwarding pipeline.

In a word, the truth-table based ACA is the most realistic approach to verify EFSs. No matter which procedure and sequence during the ACA conversion is chosen, two forwarding sets are considered as equivalent as long as they share the same truth table for same action.

3.5 Summary

With the introduction of SDN, the forwarding sets can be interpreted as a single table or multiple linked tables. Two algorithms namely MFA and ACA are discussed and verified in this chapter to convert an arbitrary table into an uniform representation called EFS. The former converts a forwarding set into a match-field-indexed dictionary in which the "key" represents "match fields" yielding a tall dictionary. The latter converts a forwarding set into an action-indexed dictionary in which the "value" represents composite "match fields" sharing the same action. It produces a fat dictionary. The experiment on Open vSwitch proves the correctness of the conversion from a table to its equivalent MFA and ACA while the performance analysis shows that ACA advances in the truth-table based expression and goes on to show that MFA is easier to implement in today's devices.

In the next chapter, a non-invasive traffic estimation application will be discussed. Compared to the equivalent forwarding set evaluation which

aims to explore the relation between FTEs, this application extends the research scope to FTEs' associated statistics.

Chapter 4

Deterministic Statistics Estimation

Software Defined Networking enables a centralised controller to monitor the network's status by collecting traffic statistics such as packets, bytes, etc. Each statistic is associated with a forwarding table entry (FTE) in a switch whose structure and format is specified by the OpenFlow standard (de-facto SDN standard). For a flow with a matching FTE, its statistic is easily acquired by an inquiry from a controller to the switch on this flow's corresponding FTE. If a flow has no matching FTE, its statistic is not known until a new FTE is installed for the purpose of monitoring it. However, the overheads of packet transmission delay and throughput loss as well as the potential conflicts between newly installed and existing FTEs jeopardise the feasibility of this approach. To avoid these drawbacks, this chapter proposes a traffic estimation approach based on the existing FTEs' statistics. With the help of Boolean algebra, the deterministic confidence interval of any given flowset can be estimated. This traffic estimation solution does not affect the existing packet forwarding functionality and performance, which makes it more practical and suitable for large-scale data-centre networks.

4.1 Research Question

In SDN, the control plane is separated from the data plane, which means the messages between switch and controller are absolutely isolated from the normal packet forwarding. The control messages are composed and decomposed in the CPU while the normal packets are processed by ASIC. As illustrated in Fig. 4.1, the control messages are sent in a dedicated control channel which is physically isolated from packet forwarding channels. An FTE manipulation based traffic estimation usually occurs in the data plane because FTE installation is conducted in ASIC, thus it alters the packet forwarding functionality and performance, which is called invasive traffic measurement. Contrary to this, the traffic estimation in the control plane does not affect the normal packet forwarding, it is a non-invasive traffic measurement.

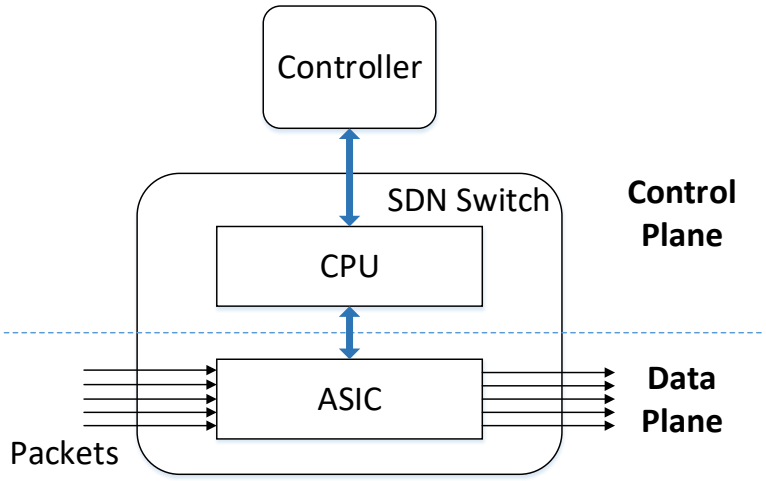


Figure 4.1: SDN architecture: control plane vs data plane

A common invasive traffic estimation is achieved by installing FTEs into a separate table and gathering the corresponding traffic statistics [95, 96]. The process is depicted in Fig. 4.2. Sometimes there exist multiple monitoring tables to serve different designated monitoring purposes and their locations also vary. Usually the monitoring tables are put in the front

of a forwarding pipeline to guarantee that all incoming traffic must pass through them [96].

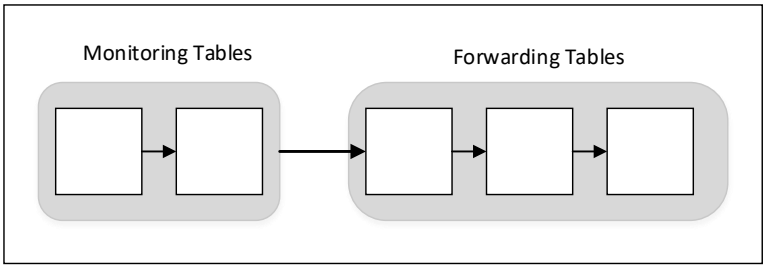


Figure 4.2: Architecture of traffic estimation by FTE installation

However, the network forwarding performance has been affected by the introduction of newly installed FTEs. Based on the research in [96], the packet forwarding throughput decreases by 1% while the latency increases by 41% on average.

Another serious problem which has been neglected in the existing research is the conflicts between monitoring flows. Here "conflict" means there are some overlaps between the monitoring flows. In this case, the match fields of their respective monitoring FTEs do not reveal their actual matching packets. This will prevent FTE with lower priority from capturing all matching packets. Two conditions must be met by the monitoring-only FTE: i) it must have the highest priority to match against all incoming packets; and ii) it must have the lowest priority to avoid having any impact on the forwarding behaviour of incoming packets. These two contradictory conditions cannot be achieved on a single-table switch which is still popular due to backward compatibility for legacy hardware. For a multi-table switch, it can be achieved by reserving the first table (Table 0) for these monitoring FTEs if the match fields of these newly installed FTEs are independent [43]. Thus all the interacted FTEs must be deployed into separate tables, which makes the FTE installation based traffic estimation less attractive because it is too expensive in terms of forwarding resources. It also makes the number of reserved monitoring tables become unpredict-

able.

To avoid these aforementioned issues, this chapter proposes a solution based on the statistics of the existing FTEs, which achieves less estimation accuracy but it is more practical and easily deployed on a large-scale network.

4.2 Non-invasive Traffic Estimation in SDN

In a software defined network, all the incoming packets are forwarded according to the rules and actions specified by the FTEs in switches; at the same time, the packet counters and bytes for this FTE are also recorded. By querying these statistics, a controller is able to estimate traffic for any given FTE's matching packets. These packets are also called flows because they always follow a certain pattern which is defined by the FTE's match fields. Typically, a flow is a sequence of packets sharing some common characteristics, for example, MAC or IP address, the same protocol, etc. However, not all flows follow the aforementioned standard flow definition. Flows can belong to an arbitrary set of flows which share some common characteristics; such an arbitrary set of flows is also called a "flowset" [84].

In this chapter, the DEterministic Confidence Interval eStimatION (DECISION) solution is proposed based on the relation between a given flowset and the existing FTEs. In this solution, there will be no new FTE installed into any switch. The deterministic confidence traffic interval for a given flowset is estimated by FTEs already installed in a switch and their associated statistics.

4.2.1 Definition

A traffic interval is a set of data with the property that any datum lying between two values in the set is also included in the set. For example, the set of all data x satisfying $0 \leq x \leq 1000$ is an interval which contains 0 and

1000 as well as all values between them. Here the traffic might be packet count, number of bytes or any units which reflect the amount of traffic in a flow. In this chapter, the statistics collected from all flow tables use the same unit (packet counter or bytes). For the estimated traffic interval, the same unit will be adopted. To simplify the expression of traffic interval, their units will be omitted in the rest of this chapter.

A traffic interval between a and b, including a and b, is denoted as $[a, b]$. When both a and b are finite and real numbers, the interval $[a, b]$ is bounded. In this case, a and b are lower bound and upper bound of this interval, respectively. For any given flowset, the traffic interval is always bounded because it must fall between 0 (no traffic at all) and the total traffic in a device. Thus, the problem of traffic interval estimation turns into finding the lower bound and upper bound for a given flowset. The absolute difference between these two bounds is also called *length* of an interval. The smaller the length of an interval is, the more accurate an estimation becomes. Thus the optimal estimated interval is determined by the greatest lower bound as well as the least upper bound.

Definition 4.2.1 (Deterministic Confidence Interval (DCI) [97])**.** A confidence interval gives an estimated range of values so defined that there is a specified probability that the value of a parameter lies within it. The estimated range is calculated from a given set of sample data. If the probability reaches 100%, it becomes a deterministic confidence interval (DCI), which means the parameter always falls in the estimated range.

Definition 4.2.2 (Optimal DCI (ODCI))**.** An optimal DCI (denoted by $\mathcal{O}$) is a DCI such that there does not exist any other DCI which is a subset of $\mathcal{O}$: $\nexists D \in \mathbb{D}, D \subset \mathcal{O}$ where $\mathbb{D}$ represents the full set of DCI for a given estimation. Thus an Optimal Deterministic Confidence Interval (ODCI) is bounded by the greatest lower bound (infimum) and the least upper bound (supremum): $\mathcal{O} = [lo, ro] : \nexists D = [lv, rv] \in \mathbb{D}, lo < lv, ro > rv$. They are denoted as the lower bound of ODCI (LODCI) and the upper bound of ODCI (UODCI), respectively.

With the definition of ODCI, the DECISION approach turns into the problem of finding ODCI for a given flowset.

4.2.2 Problem Description

A flowset is an arbitrary set of flows defined by the specified common fields (SCF), for example, a flowset f whose medium access control (MAC) address equals "00:11:22:33:44:55" ($SCF_f = \{MAC = 00{:}11{:}22{:}33{:}44{:}55\}$) or a combination of a virtual LAN (VLAN) ID 1 and a Multiprotocol Label Switching (MPLS) label 101 ($SCF_f = \{VLAN = 1, MPLS = 101\}$). In this chapter, SCF and an FTE's match fields are expressed in a form of Boolean function, i.e., $SCF = g(a_1, a_2, \ldots, a_n)$ where a_i represents the attributes of a flowset which has been specified in latest OpenFlow standard [90] and function g represents a Boolean logical operation (e.g., "AND($\wedge$)", "OR($\vee$))", "NOT($\neg$)", etc.) on these attributes. With the introduction of SCF, an ODCI estimation problem can be addressed with the help of set relation between SCF and the match fields of a forwarding set. The formal definition of this problem is expressed as follows:

Problem 4.2.1 (**Optimal DCI Estimation**). *Given a forwarding set F and a flowset f which is defined by its specific common fields SCF_f, the ODCI estimation problem is to identify the traffic interval of f (denoted as T_f) whose length is as small as possible. Here F is composed of forwarding table entries in the form of single-table or multi-table, each entry is associated with a statistic. Specifically, the traffic interval is composed of lower bound l and upper bound u, i.e., $T_f = \big[l, u\big]$. Thus the ODCI estimation problem turns into the maximisation of l and the minimisation of u.*

For a switch, all the FTEs, routing entries, access control lists (ACL) and high level polices determine the forwarding behaviour of all incoming packets, these various types of forwarding entries are called *forwarding set*. In SDN, a forwarding set is interpreted in the form of a single table or

chained multiple tables to enable packets processing in a more sophisticated way. The OpenFlow standard specifies the structure of a forwarding table which is illustrated in the top half of Fig. 4.3 [90]. The bottom half of Fig. 4.3 shows a detailed multi-table structure with three linked tables. These tables collectively form a forwarding set to match against all incoming packets and execute associated actions. Since these chained tables behave as a packet processing pipeline, they are also called *forwarding pipeline*. In this forwarding pipeline, the actions in "Table 0" contain "Goto Table" attribute which means the matched packets of this forwarding entry will be passed to another table for further processing. A packet with attributes "A" and "B" will be sent to Table 1 and Table 2 for further processing, respectively. Note that in each table, every single entry is associated with a statistic, which means a packet's traffic will be counted more than one time if it travels along multiple tables.

Our DECISION solution in section 4.3 begins from a single forwarding table and then extends to the multi-table scenario. A single flow table is composed of multiple flow entries which is illustrated as a row in a table. Take the table in the top half of Fig. 4.4 as an example, there are a total of three FTEs in which "Match fields", "Action" and "Priority" are OpenFlow's intrinsic attributes. While each FTE is always associated with a statistic, the "Statistics" is not part of an FTE. For the sake of easy reference to a specific FTE, an "Index" column is also added into this table.

Using the forwarding table example in Fig. 4.4 which contains three FTEs indexed 1-3 (the top half) as shown, the ODCI for the flowset with SCF as $\{MAC = A, IP = B\}$, $\{IP = B\}$ and $\{MAC = D, IP = B\}$ are $[40, 40]$, $[100, 180]$ and $[0, 60]$, respectively (bottom half in Fig. 4.4). Take the second SCF $\{IP = B\}$ as an example, the packets with IP address as B may match FTE 1 or 3, thus the LODCI is the sum of FTE 1's and FTE 3's statistics ($40 + 60 = 100$). However, the packets with MAC address as C and IP address as B will only match FTE 2, which means the UODCI of SCF $\{IP = B\}$ is the sum of all the three FTE's statistics in this table

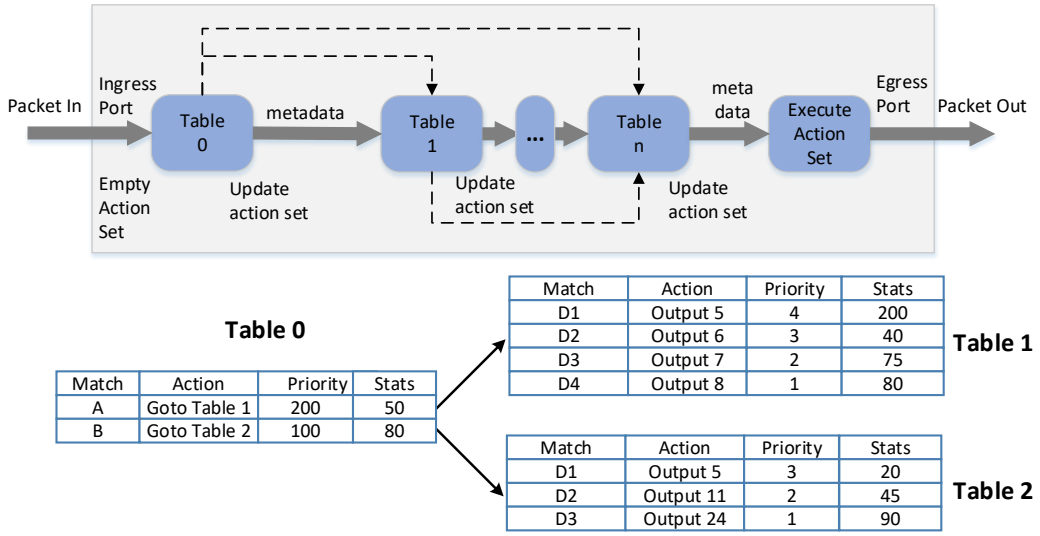


Figure 4.3: Forwarding table structure: a typical OpenFlow multiple flow table forwarding pipeline specified in OpenFlow specification (on top) and an example with three tables in which Table 0 has the "Goto Table" action (on bottom)

$(40 + 80 + 60 = 180)$. In this case, a packet with IP address as B may hit any of these three FTEs.

4.3 Deterministic Traffic Estimation Algorithms

4.3.1 Single-Table Traffic Statistics Estimation

Intuitively, LODCI of a given flowset f is the sum of the statistics for all FTEs whose match-field set is a subset of f's SCF (denoted as SCF_f). The UODCI, however, is a set cover problem i.e. to find a collection of FTEs whose union match-field set is a superset of SCF_f. This approach is denoted as Simple-Subset-Superset (S^3) in the reminder of this chapter. Take the forwarding table in Fig. 4.5 as an example, the match fields of the first two FTEs M_1 and M_2 are subsets of flowset f , thus the lower bound of flowset f is the sum of these two FTEs' statistics (i.e. 120). The union of all

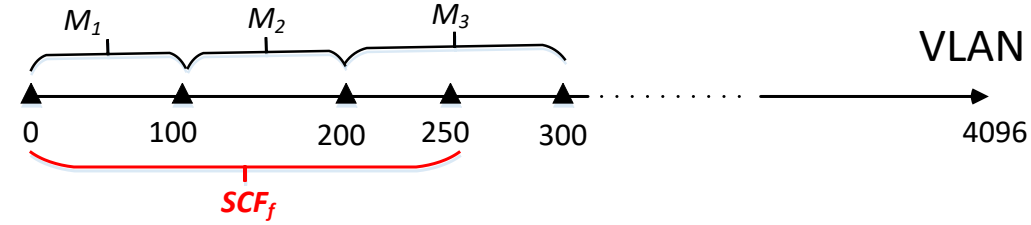
Index	Match Fields	Action	Priority	Statistics
1	MAC = A, IP = B	Port 1	200	40
2	MAC = C	Port 4	90	80
3	IP = B	Port 12	70	60

SCF_f	Lower bound	Upper bound
MAC = A, IP = B	40	40
IP = B	100	180
MAC = D, IP = B	0	60

Figure 4.4: A single forwarding table ODCI estimation example

the three sets $\{M_1, M_2, M_3\}$ is a superset of SCF_f, thus the upper bound of flowset f is the sum of all the statistics (i.e. 180).

Index	Match Fields	Action	Priority	Statistics
1	VLAN = [1 – 100]	Port 1	200	40
2	VLAN = [101 – 200]	Port 4	90	80
3	VLAN = [201 – 300]	Port 12	70	60



SCF_f	Low bound	Upper bound
VLAN = [1-250]	40+80	40+80+60

Figure 4.5: A traffic estimation example on independent FTE set

In the example illustrated in Fig. 4.5, all match fields are independent, (i.e. their match fields have no intersections). However, for FTEs with intersections, the S^3 approach does not guarantee that an optimal lower and upper bound can be found. Figure 4.6 demonstrates a forwarding table with two FTEs (Index 1 and 2) whose match fields intersect. For a flowset $SCF_f = \{$ VLAN ranges from 11 to 150$\}$, there is no FTE whose

match-field set is the subset of f and the estimate is always 0. For the upper bound, the minimum covering set of f are FTE 2 and 3's match fields, thus the upper bound is the sum of these two FTEs' statistics (i.e. $80 + 60 = 140$).

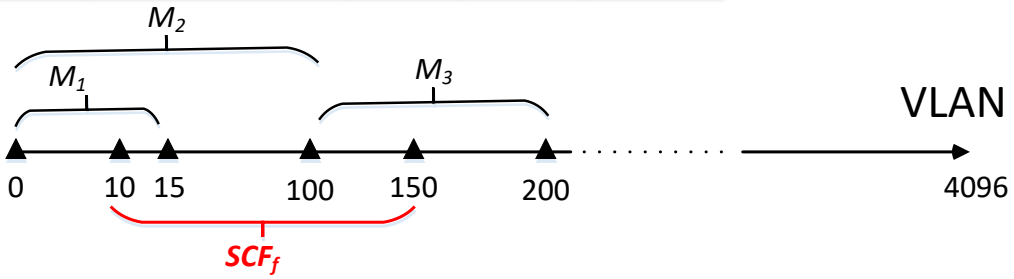
Unfortunately, the ODCI estimation based on the S^3 approach for the case in Fig. 4.6 is not correct. As mentioned earlier, all incoming packets to a switch match against the FTEs in a single table in the sequence of decreasing priority. Once a packet matches an FTE, the corresponding action will be performed and the remaining FTEs will be ignored. As an example, a packet with VLAN between 1 and 15 will never match against FTE 2 in Fig. 4.6, thus the effective VLAN id of FTE 2 ranges from 16 to 100 rather than from 1 to 100. Hence all FTE 2's matching packets belong to the estimated flowset whose $SCF_f = \{VLAN = [11, 150]\}$ and the lower bound of flowset f should be 80 rather than 0. Similarly, the union set of the FTE 2 and 3's match fields do not include VLAN id between 11 and 15 (inclusive) and it is not the cover set of the given flowset's SCF. The correct LODCI and UODCI of this given flowset should be [80, 180] rather than [0, 140].

Because the proposed S^3 approach is only applicable to the independent FTE sets, the intersections between two FTEs' match fields must be eliminated. The elimination will lead to a large match set for each FTE and thus it is unlikely to be adopted in the real-world traffic estimation. However, it is a rigorous approach and the ODCI will be guaranteed. Besides this, a pragmatic approach which does not require removing dependencies on all FTEs is also proposed. It is more practical because it offers a coarse traffic estimate involving far less computation.

4.3.1.1 AS^3 Approach

In a single OpenFlow table, an incoming packet will hit only one FTE no matter how many FTEs it matches. Therefore an FTE's **actual** match-field (AMF) set excludes all the match fields with higher priorities. Once an

Index	Match Fields	Action	Priority	Statistics
1	VLAN = [1 – 15]	Port 1	200	40
2	VLAN = [1 – 100]	Port 4	90	80
3	VLAN = [101 – 200]	Port 12	70	60



SCF_f	Low bound	Upper bound
VLAN = [11-150]	0 ?	80+60 ?

Figure 4.6: A traffic estimation example on non-independent FTE set

FTE's original match-field (OMF) set is replaced by its AMF set, these FTEs are independent (within a table) and their positions in a table are exchangeable. This process is called "deprioritisation" because the "Priority" attribute is safely removed without changing forwarding behaviour. In the following discussion, the S^3 approach will be called AS^3 when applied on AMF and OS^3 when applied on the original match fields.

Fig. 4.7 illustrates the deprioritisation process with a five-entries example. In Table 1 (the original table), each item is associated with a priority which determines the matching sequence of an incoming packet. To eliminate the dependencies among different FTEs, a virtual attribute "Unmatch Fields" is added in each FTE to exclude the match fields in all FTEs with higher priorities. Thus an FTE's AMF can be expressed by a logical *AND* operation between the original "Match Fields" and the newly added "Unmatch Fields". Here "Unmatch Fields" is the logical negation of the union of all previous FTEs' OMFs. The deprioritisation preprocessing of a single table is depicted in Algorithm 7 where "$\sum$" represents the logical

operation "OR" ($\vee$) and "$\cdot$" means the logical operation "AND" ($\wedge$).

Table 1

Index	Match Fields	Action	Priority	Statistics
1	M_1	A_1	200	S_1
2	M_2	A_2	90	S_2
3	M_3	A_3	70	S_3
...	...	...	...	...
N	M_N	A_N	10	S_N

Table 2

Index	Match Fields	Unmatch Fields	Action	Priority	Statistics
1	M_1	NULL	A_1	X	S_1
2	M_2	M_1	A_2	X	S_2
3	M_3	M_1 M_2	A_3	X	S_3
...	...	...	...	X	...
N	M_N	M_1 M_2... M_{N-1}	A_N	X	S_N

Figure 4.7: Single table deprioritisation

In a deprioritised forwarding table, the original match field of an FTE has been converted into AMF and the AMFs in a table are independent. Algorithm 8 outlines the process of computing the LODCI (lv) and UODCI (rv) from a deprioritised table using AS^3. The LODCI is the sum of the statistics ($stat$) for all FTEs whose match-field set is a subset of the given flowset SCF (Scf) (Algorithm 8, lines 7-9). For UODCI, all FTEs are enumerated to find the SCF's intersected FTEs, the sum of these FTEs' associated statistics is UODCI (Algorithm 8, lines 10-15). If the default FTE (packets sent to controller or dropped) is taken into consideration, the union of all FTEs' match fields is a full set of SCF, thus the UODCI is finite and computable.

Algorithm 7 Single table preprocessing: deprioritisation

1: $T \leftarrow$ input/output: An Array of FTEs with Match Fields (input)
2: and it will be converted to an array of FTEs with AMF (output)
3: **procedure** $Deprioritisation(T)$
4: $T[0].amf \leftarrow T[0].omf$
5: **for** $i = 1,\ \ i < T.count,\ \ i{+}{+}$
6: $T[i].amf \leftarrow T[i].omf \cdot (\neg(\sum_{j=1}^{j=i-1} T[j].omf))$
7: **end for**
8: **end procedure**

4.3.1.2 OS^3 Approach

The operations on SCF and FTEs' match fields are Boolean logical operation upon their respective Boolean functions. The complexity of a Boolean function depends on two factors: width and length. The width means number of literals in a formula-based Boolean function or the bits of a value in a truth-table based Boolean function while the length represents the number of products in a disjunctive normal form (DNF) or values in a truth table [92].

In deprioritisation, an FTE's AMF is composed of this FTE's original match fields and the compliment of the AMF of the FTEs with higher priority. The length of an AMF is usually far greater than the original match-field set. This introduces a significant computation effort when performing the Boolean operations on AMF. A more practical approach with DECISION builds ODCI without deprioritisation.

In OS^3, a given flowset f's statistics must not be less than the sum of all FTE's statistics if these FTE's match-field sets are subsets of f's SCF. It means a flowset's lower bound is still able to be estimated based on the subset relation between FTE's match fields and a flowset's SCF. However, for the case that an FTE's AMF rather than its OMF is a subset of a given flowset, this FTE will be excluded from the computation of the lower bound. Referring back to Fig. 4.6, the second FTE cannot be included in

Algorithm 8 Single table ODCI estimation (AS^3)

1: $Scf \leftarrow$ input: SCF of a given flowset
2: $T \leftarrow$ input: An Array of FTE with AMF
3: *lv, rv* $\leftarrow$ lv: LODCI, rv : UODCI, $[lv, rv]$ is the ODCI
4: **procedure** $Single\_table\_ODCI(Scf, T, lv, rv)$
5: $lv \leftarrow 0; rv \leftarrow 0$; $S[0] \leftarrow Scf$
6: **for** $i = 1,\ \ i <= n,\ \ i{+}{+}$
7: **if** $T[i].amf \subseteq Scf$ **then**
8: $lv \leftarrow lv + T[i].stat$
9: **end if**
10: **if** $S[i] \neq \emptyset$ **then**
11: $S[i] \leftarrow S[i-1] \cap (\neg T[i].amf)$
12: **if** $(T[i].amf \cap S[i-1]) \neq \emptyset$ **then**
13: $rv \leftarrow rv + T[i].stat$
14: **end if**
15: **end if**
16: **end for**
17: **return** lv, rv
18: **end procedure**

the computation of lower bound because its original VLAN range is not a subset of the given SCF_f while its AMF is subsumed into SCF_f. Thus, the lower bound achieved by OS^3 is lower than AS^3, which means that OS^3 cannot provide an optimal DCI in some circumstances. Although the AS^3 approach achieves a better (greater) lower bound than OS^3, both achieve the same upper bound estimates, and this is given as a proof below:

Proof. The proof of the equivalence of UODCI between AMF and OMF relies on two conditions which must be met in each iteration : i) the value of $S[i]$ is identical for both AMF and OMF in step i; ii) the conditional statement of $(T[i].amf \cap S[i-1]) \neq \emptyset$ (line 12 in Algorithm 8) yields identical result (i.e. *true* or *false*) if $T[i].amf$ replaces $T[i].omf$.

By definition, there exists $T[0].amf = T[0].omf$. By induction on k, it

has the same $S[k]$: $S_{amf}[k] = S_{omf}[k]$. Let $i = k + 1$, then:

$$S_{amf}[k+1] = S[k] \cap (\neg T[k+1].amf)) \tag{4.1}$$

$$\begin{aligned} T[k+1].amf =& T[k+1].omf \cap (\neg T[0].omf) \\ & \cap (\neg T[1].omf) \cdots \cap (\neg T[k].omf) \end{aligned} \tag{4.2}$$

where Eq. 4.1 and Eq. 4.2 come from Algorithm 8 Line 11 and Algorithm 7 Line 6, respectively. For the case of OMF, then:

$$S[k] \cap T[i].omf = \emptyset \ \forall i \in [0, k], \tag{4.3}$$

which holds because in the $(k-1)$-th iteration, it can be asserted that $S[k] = S[k-1] \cap (\neg T[k].omf))$. The right hand side of Eq. 4.1 is expanded based on Eq. 4.2 to give:

$$\begin{aligned} & (S[k] \cap T[k+1].omf) \cup (S[k] \cap T[0].omf) \\ & \cup (S[k] \cap T[1].omf) \cdots \cup (S[k] \cap T[k].omf), \end{aligned} \tag{4.4}$$

and then simplified to:

$$S[k] \cap T[k+1].omf, \tag{4.5}$$

which is by definition $S_{omf}[k+1]$. Therefore, substituting Eq. 4.5 back to Eq. 4.1 yields to desired result:

$$S_{amf}[k+1] = S_{omf}[k+1]. \tag{4.6}$$

Similarly, it also holds for the second condition: $T[i].amf \cap S[i-1].amf = T[i].omf \cap S[i-1].omf$ using Eq. 4.2-4.5. Thus, Algorithm 8 produces identical UODCI with AMF and OMF. □

4.3.2 Multi-Table Traffic Statistics Estimation

All the FTEs in a switch are constructed either in the form of single table or multiple tables. For a multi-table switch, all packets must match against "Table 0" for subsequent forwarding to other tables via the "Goto Table" action. As illustrated in Fig. 4.8, a packet has potentially multiple paths to reach its egress port(s). Hence the estimation of a flowset must consider all the potential paths to compute the ODCI. In this example, an arrow line is added between two tables if there exists one FTE in the former table which contains a "Goto Table" action to the next table. For a point in a forwarding pipeline, potential paths for a packet passing through this point can be identified by the corresponding table indices. For example, for any point between "Table 0" and "Table 1", the potential paths for a packet are $\{T_{01}, T_{02}, T_{03}, \}$ in which T_{ij} represents a path between "Table i" and "Table j". To take the packets going straightforward from one table to the egress port(s) into consideration, a virtual end point "5" is added. Thus, T_{45} represents the path between "Table 4" and the egress ports, the traffic passing through it includes all packets egressing from physical ports.

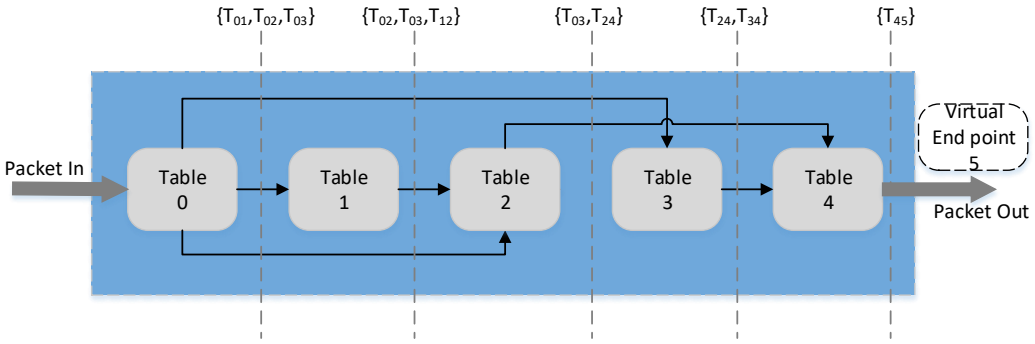


Figure 4.8: An example of multi-Table ODCI estimation

At any given point in a forwarding pipeline, the local ODCI is an accumulation of the statistics for all the possible forwarding paths at the same point. Then a global ODCI can be easily selected from these local ODCIs. The process is demonstrated in Algorithm 9 in which $S_{ij} = [lv_{ij}, rv_{ij}]$ rep-

resents the local ODCI between Table i and j. In Algorithm 9, it is assumed that the ODCI for each table has been computed. Thus, a local estimation of a point between two tables is achieved by accumulating the statistics of all the potential paths at that point (Algorithm 9 Lines 6-12). Finally a global optimal estimation is accomplished by selecting the greatest LODCI and least UODCI (Algorithm 9 Lines 13-18).

Algorithm 9 Multiple table ODCI traffic estimation

1: $S_{ij} = [lv_{ij}, rv_{ij}] \leftarrow$ input: local ODCI
2: $N \leftarrow$ input: index of the last table $(0 \leq i \leq N)$
3: *lg, rg* $\leftarrow lg$: global LODCI, rg: global UODCI
4: **procedure** $Multiple\_table\_ODCI(S, N, lg, rg)$
5: **for** $k = 0,\ \ k \leq N,\ \ k{+}{+}$
6: $tmp\_lg \leftarrow 0\ ,\ tmp\_rg \leftarrow 0$
7: **for** $i = 0,\ \ i \leq k,\ \ i{+}{+}$
8: **for** $j = i + 1,\ \ j \leq N + 1,\ \ j{+}{+}$
9: $tmp\_lg \leftarrow tmp\_lg + lv_{kj}$
10: $tmp\_rg \leftarrow tmp\_rg + rv_{kj}$
11: **end for**
12: **end for**
13: **if** $tmp\_lg > lg$ **then**
14: $lg \leftarrow tmp\_lg$
15: **end if**
16: **if** $tmp\_rg < rg$ **then**
17: $rg \leftarrow tmp\_rg$
18: **end if**
19: **end for**
20: **return** lg, rg
21: **end procedure**

4.4 Evaluation

The evaluation of the DECISION solution was performed by comparing the statistics measured on the traffic generator (ground truth) and the stat-

istics estimated on SDN controller. The traffic tests will be executed on different scales of number of tables and number of FTEs.

4.4.1 Functional Test

4.4.1.1 Test Bed & Traffic

An evaluation of the DECISION solution was performed on an Open vSwitch (version 2.6.1) which supports 255 flow tables [93]. As illustrated in Fig. 4.9, three virtual machines (VMs) on a Ubuntu server are created, one is installed with an Open vSwitch as OpenFlow switch (vm2) and the rest two VMs as traffic generator (vm1) and receiver (vm3). On the host OS, Ryu (version 4.11) [98] is installed as a controller to provision OpenFlow rules and retrieve the traffic statistics while Scapy [94] is the traffic generator. More details about this test bed are listed in Appendix D.

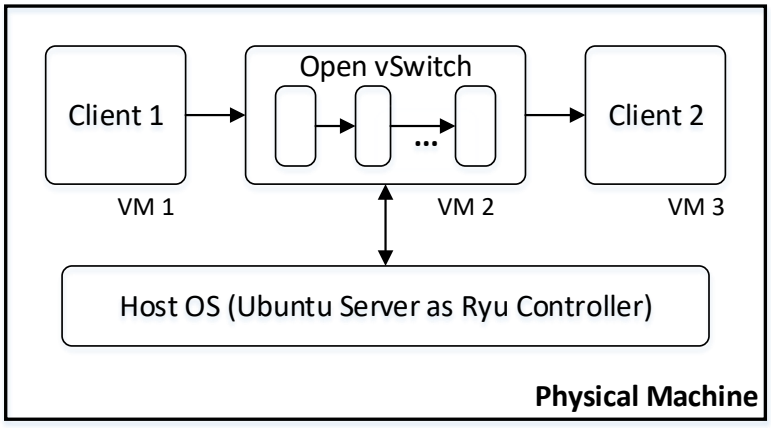


Figure 4.9: Test bed environment

In this evaluation, the estimated statistic for a given flowset is compared with the ground truth, the actual traffic, which is measured by installing a new FTE with the exact same matching fields. This is achieved by reserving tables in the head of a forwarding pipeline for traffic measurement only [43]. In the following description, the new installed FTE

and reserved table are called "traffic measurement FTE" (TM-FTE) and "traffic measurement table" (TM-Table), respectively. Figure 4.10 depicts the design where the first table (or tables) is reserved for measurement and the rest behaves as normal packet forwarding.

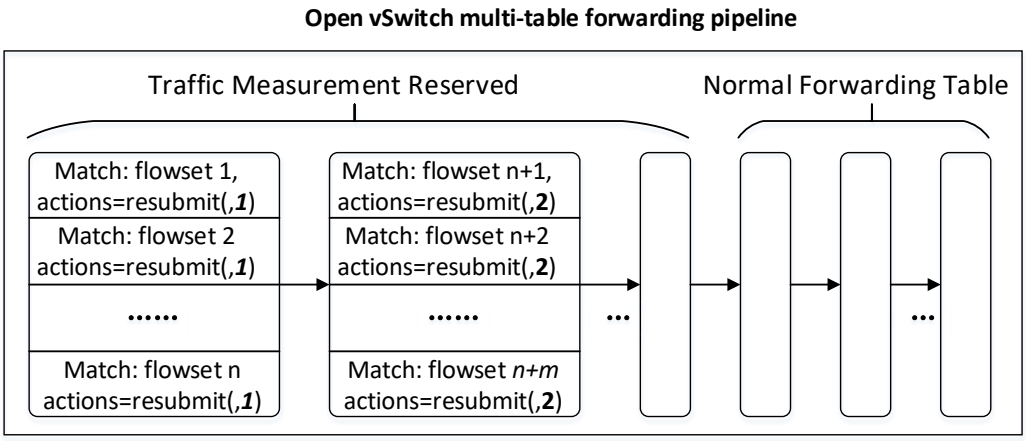


Figure 4.10: Reserving tables for traffic measurement

4.4.1.2 Functional Test Result & Analysis

In the DECISION functional testing, both correctness and accuracy are evaluated. A correct traffic estimation means the actual traffic value (av) falls into the range of the estimated ODCI: $lo \leq av \leq ro$ where lo and ro stand for LODCI and UODCI, respectively.

The accuracy of ODCI reflects how close an estimation is to the actual traffic. In the DECISION approach, accuracy is determined by the distance between LODCI and ground truth (the actual traffic, denoted as av in Eq. 4.7) as well as UODCI and ground truth. In each estimation, the sum of aforementioned distances must be always less than the total traffic passing through a switch, herein a relative distance is chosen to measure the accuracy among different estimations. As expressed in Eq. 4.7, the accuracy is determined by the relative length of ODCI.

$$
\begin{aligned}
accuracy &= \left\{1 - \frac{|av - lo| + |ro - av|}{total}\right\} \times 100\% \\
&= \left\{1 - \frac{|ro - lo|}{total}\right\} \times 100\%.
\end{aligned}
\tag{4.7}
$$

Table 4.1: Single table, single non-wildcarded match field functional test result

FTE	Generated Traffic Pattern	FTE Count	Total Traffic (Bytes)	Ground Truth (Bytes)	LODCI (Bytes)	UODCI (Bytes)	Accuracy (95% CI)
VLAN	Dot1Q(vlan=RandNum(1,1024))	2^9	$6.55 * 10^6$	$4.96 * 10^3$	$3.92 * 10^2$	$3.18 * 10^6$	$51.5\% \pm 8.2\%$
VLAN	Dot1Q(vlan=RandNum(1,1024))	2^8	$6.55 * 10^6$	$3.92 * 10^3$	$3.84 * 10^2$	$4.97 * 10^6$	$24.1\% \pm 4.5\%$
VLAN	Dot1Q(vlan=RandNum(1,1024))	2^7	$6.55 * 10^6$	$3.84 * 10^3$	$2.56 * 10^2$	$5.64 * 10^6$	$13.9\% \pm 2.1\%$
VLAN	Dot1Q(vlan=RandNum(1,1024))	2^6	$6.55 * 10^6$	$3.20 * 10^3$	$1.28 * 10^2$	$6.11 * 10^6$	$6.7\% \pm 1.1\%$

Table 4.2: Single table, single wildcarded match field functional test result

FTE	Generated Traffic Pattern	FTE Count	Total Traffic (Bytes)	Ground Truth (Bytes)	LODCI (Bytes)	UODCI (Bytes)	Accuracy (95% CI)
DIP	IP(dst=RandIP("10.1.*.*"))	2^{15}	$6.55 * 10^6$	$4.96 * 10^3$	$3.92 * 10^2$	$1.82 * 10^6$	$72.2\% \pm 9.7\%$
DIP	IP(dst=RandIP("10.1.*.*"))	2^{14}	$6.55 * 10^6$	$3.92 * 10^3$	$3.84 * 10^2$	$3.44 * 10^6$	$47.4\% \pm 7.4\%$
DIP	IP(dst=RandIP("10.1.*.*"))	2^{13}	$6.55 * 10^6$	$3.84 * 10^3$	$3.20 * 10^2$	$4.12 * 10^6$	$37.1\% \pm 6.3\%$
DIP	IP(dst=RandIP("10.1.*.*"))	2^{12}	$6.55 * 10^6$	$3.20 * 10^3$	$2.56 * 10^2$	$4.88 * 10^6$	$25.5\% \pm 5.1\%$
DIP	IP(dst=RandIP("10.1.*.*"))	2^{11}	$6.55 * 10^6$	$1.92 * 10^3$	$1.28 * 10^2$	$5.18 * 10^6$	$20.9\% \pm 3.9\%$
DIP	IP(dst=RandIP("10.1.*.*"))	2^{10}	$6.55 * 10^6$	$2.56 * 10^2$	$6.4 * 10^1$	$5.78 * 10^6$	$11.7\% \pm 2.7\%$

The functional test results in single non-wildcarded match field (VLAN), single wildcarded match field (DIP) and multiple match fields (DMAC + DIP) for single table scenario are presented in Table 4.1, 4.2 and 4.3, respectively. Here "wildcard" indicates whether a match field contains the "ANY" bit (the bit will match both 0 and 1, usually denoted as an asterisk (*)). In these tables, the columns "FTE", "Traffic Pattern", and "FTE Count" denote the FTE's match fields, the pattern of the traffic generated by Scapy, and the number of FTEs in the table, respectively. The fifth

Table 4.3: Single table, multiple match fields functional test result

FTE	Generated Traffic Pattern	FTE Count	Total Traffic (Bytes)	Ground Truth (Bytes)	LODCI (Bytes)	UODCI (Bytes)	Accuracy (95% CI)
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{19}	$6.55 * 10^6$	$4.96 * 10^3$	$1.28 * 10^3$	$3.83 * 10^6$	$41.5\% \pm 6.7\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{18}	$6.55 * 10^6$	$3.92 * 10^3$	$4.96 * 10^2$	$4.21 * 10^6$	$35.7\% \pm 6.1\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{17}	$6.55 * 10^6$	$3.84 * 10^3$	$4.96 * 10^2$	$4.49 * 10^6$	$31.5\% \pm 5.8\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{16}	$6.55 * 10^6$	$3.84 * 10^3$	$3.20 * 10^2$	$4.83 * 10^6$	$26.3\% \pm 5.2\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{15}	$6.55 * 10^6$	$1.28 * 10^3$	$1.92 * 10^2$	$5.11 * 10^6$	$22.0\% \pm 3.8\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{14}	$6.55 * 10^6$	$1.92 * 10^2$	$6.4 * 10^1$	$5.48 * 10^6$	$16.3\% \pm 3.1\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{13}	$6.55 * 10^6$	$1.28 * 10^2$	$6.4 * 10^1$	$6.02 * 10^6$	$8.1\% \pm 2.2\%$

column, "Total Traffic", is the recorded traffic passing through the table for each test, i.e., the value of $total$ in Eq. 4.7. The "Ground Truth" is the actual traffic (av in Eq. 4.7) which is measured by TM-FTE (Fig. 4.10). Each set of tests is repeated ten times and the average accuracy is computed and shown in the last column.

From the observation of all the values in the column of "Ground Truth" in all three tables, the ground truth always falls in the range [LODCI, UODCI], which means the traffic estimation results are correct. For each test scenario, various numbers of FTEs (denoted by column "FTE Count") are installed into Open vSwitch to verify the correlation between the number of FTEs and the estimation accuracy. The accuracy increases with increasing number of FTEs because more FTEs are likely to produce a closer simple-subset-superset that matches a given flowset's SCF.

In Table 4.2, the traffic patten has $2^{16} - 2$ valid variations. It can be observed that the accuracy grows when the number of FTEs increases (the match fields *base* held constant). Here the base is the total number of possible combinations for a given type of match field. In this table, the base of the traffic is $2^{16} - 2$, thus the estimation accuracy with 2^{15} FTEs (72.2%)

will be better than the accuracy with 2^{10} FTEs (11.7%) for the same traffic pattern "10.1.*.*" because the probability of finding a closer superset and subset in the former case is higher.

Table 4.4: Multi-table, multiple match fields functional test result

FTE	Generated Traffic Pattern	FTE Count	Total Traffic (Bytes)	Ground Truth (Bytes)	LODCI (Bytes)	UODCI (Bytes)	Accuracy (95% CI)
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{19}	$6.55 * 10^6$	$4.96 * 10^3$	$3.92 * 10^2$	$4.04 * 10^6$	$38.3\% \pm 6.2\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{18}	$6.55 * 10^6$	$3.92 * 10^3$	$3.84 * 10^2$	$4.81 * 10^6$	$26.5\% \pm 5.3\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{17}	$6.55 * 10^6$	$3.20 * 10^3$	$3.84 * 10^2$	$5.34 * 10^6$	$18.4\% \pm 3.7\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{16}	$6.55 * 10^6$	$2.56 * 10^3$	$1.92 * 10^2$	$5.66 * 10^6$	$13.6\% \pm 3.1\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{15}	$6.55 * 10^6$	$1.92 * 10^3$	$1.28 * 10^2$	$5.93 * 10^6$	$9.4\% \pm 2.7\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{14}	$6.55 * 10^6$	$1.92 * 10^2$	$6.4 * 10^1$	$6.07 * 10^6$	$7.3\% \pm 2.4\%$
DMAC DIP	Ether(dst=RandMAC("11:22:33:44:*:*")) IP(dst=RandIP("10.1.1.*"))	2^{13}	$6.55 * 10^6$	$1.28 * 10^2$	$6.4 * 10^1$	$6.11 * 10^6$	$6.7\% \pm 2.1\%$

In the functional test results illustrated in Table 4.1, 4.2 and 4.3, the accuracies vary from 6.7% to 72.2%. The low accuracies in these experiments are caused by the randomly generated match fields of estimated flowsets. In typical traffic estimation cases, such scenarios are unlikely to occur because the flowset to be estimated is usually highly correlated with existing FTEs and are more likely to have narrower UODCI and LODCI than the randomly selected flowset.

The estimation accuracies increase when increasing the number of FTEs, and this is easily observed in Fig. 4.11 with the experiment on an IP match field whose first 16 bits are fixed. In other words, its base size is 2^{16}. The ratio of the number of FTEs to the base explains why the accuracies of the first two tests in both tables are very low ($\leq 20\%$). A low ratio means a low probability to find the appropriate subsets and supersets for a given flowset. All these experiments clearly indicate a high positive correlation

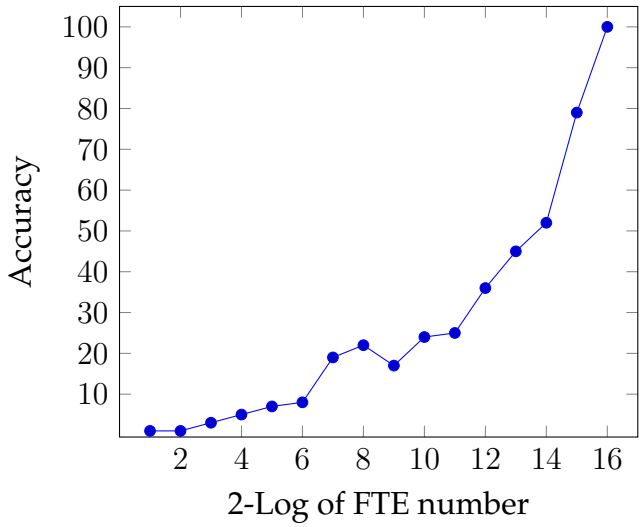


Figure 4.11: Increasing accuracy with number of FTEs

between the number of FTEs and accuracy.

The same test cases in Table 4.3 are also repeated on a multi-table scenario in which the same number of FTEs are spanned in five tables randomly. Table 4.4 demonstrates the estimation accuracies, each of them is slighter better than the corresponding case in Table 4.3. This is mainly attributed to the greatest LODCI and least UODCI selection process in multiple table traffic estimation (Algorithm 9).

All the test results in Table 4.1, 4.2, 4.3 and 4.4 indicate that the estimation accuracy is determined by the ratio of the number of residing FTEs to the base of match fields. Actually this accuracy will be improved in real production environment in which the monitoring flowset is usually highly correlated with the already installed FTEs. Moreover, it is also more suitable to large-scale networks such as data-centres because the accuracy will also benefit from large quantities of FTEs.

4.4.2 Performance Evaluation

The performance evaluation will be conducted from four perspectives on both DECISION and traditional FTE installation based approaches. They

are detection time and bandwidth overhead in control plane, the impacts on packet transmission delay and throughput in data plane.

4.4.2.1 Control Plane Performance in DECISION

In DECISION, the time to complete a round of traffic estimation can be expressed as:

$$T_{est} = T_{comm} + T_{switch} + T_{controller} \ ,$$

where T_{comm} represents the transmission time between controller and switches, T_{switch} denotes the querying time for traffic statistics and $T_{controller}$ denotes the time to process traffic statistics at the controller side. In this experiment, the transmission time and statistics querying time on switch side is calculated from the timestamp in multipart request message and multipart response message. The format of multipart message is defined in the OpenFlow specification to retrieve flow table statistics. Since it is hard to measure the transmission time and statistics querying time separately, they are measured as a single value which is illustrated in Fig. 4.12. It can be observed from this figure that the retrieval time increases nearly linearly to the number of FTEs. After observing the multipart format in Fig. 4.14, every pair of multipart request and response messages only carries a certain number of FTEs statistics, this means the number of multipart messages are proportional to the number of FTEs. It also explains the linear relationship between statistics retrieval time and the scale of FTE.

The processing time at the controller, $T_{controller}$, is also proportional to the number of FTEs. Figure 4.13 demonstrates the computation time of various scales of FTEs. The tests are conducted on a Dell OptiPlex 9020 server with Intel i7 Quad-Core CPU and 8GB RAM. The experiment shows that the computation time is not sensitive to the width of match fields, which means the number of FTEs rather than the type of FTE determines the computation time on controller. The most time-consuming computa-

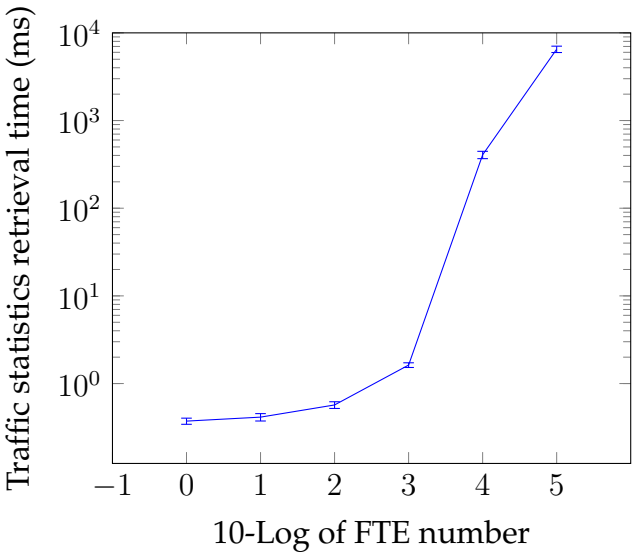


Figure 4.12: Traffic statistics retrieval time with increasing number of FTEs (95% CI)

tion on controller is the Boolean operations among different FTEs' match fields, which is loaded into memory and computed simultaneously regardless of the width of the Boolean values.

Based on the measurements in Fig. 4.12, suppose there are 10^5 FTEs in a flow table, the maximum time for statistics retrieval and computation is around 6,510 ms and 86 ms, respectively. Thus, the major time overhead lies in the traffic statistics retrieval rather than the computation on controller. In this experiment, the traffic estimation on an Open vSwitch with 10,000 flows and an entry-level server takes around six seconds to complete.

The performance evaluations on bandwidth are also conducted on the same scale of FTEs ranging from 1 flow to 100,000 flows. Figure 4.15 illustrates the bandwidth overhead of flow status which is near-linear to the number of FTEs. It can be explained by the format of multipart OpenFlow message. As demonstrated in Fig. 4.14, besides the fixed length of IP header, every reply message contains a certain number flows' status. Thus the traffic of FTE statistics is mainly determined by the number of

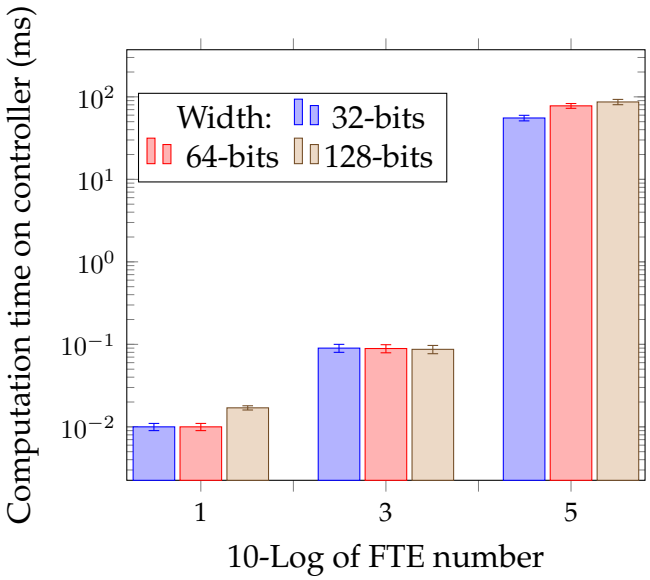


Figure 4.13: Traffic statistics computation time with increasing number of FTEs (95% CI)

FTEs and the size of each flow status. In this experiment, all FTEs are constructed with a random MAC address as matching fields, thus they have the same constant flow status length: 88 bytes. The total traffic overhead for an Open vSwitch table with 100,000 flows is around 100,000*88 bytes = 8.8 Mbyte, which is acceptable for any control plane transmission media. In a small scale network, control plane relies on a switch's management port whose bandwidth is either 10Mbps or 1000Mbps. The bandwidth can be increased to 10Gbps in large-scale network where one or more aggregated data ports are allowed to be reserved for the control plane. From this perspective, the bandwidth in the control plane will not become the bottleneck of DECISION. Thus the following discussion will mainly focus on the reduction of traffic estimation time.

There are two methods to improve the efficiency of traffic statistics retrieval. In real scenario, the process of reading all flows' statistics is usually divided to multiple stages. For example, firstly, a controller asks all switches to report table-level rather than individual flows' status, and then

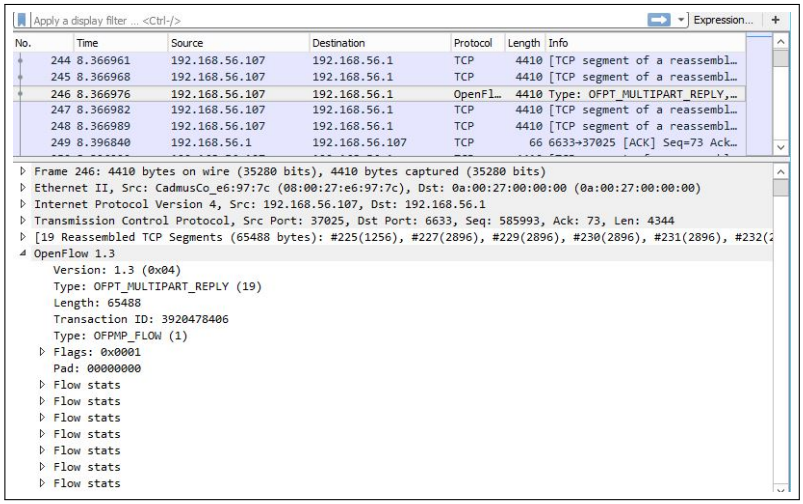


Figure 4.14: OpenFlow multipart message format

the controller selects interested flow table(s) and gets their aggregated statistics. In the final stage, a small number of individual flows' statistics will be collected and analysed. The process is fully explained in section 5.2.2.

This coarse-to-fine multi-stage approach reduces the number of monitoring flows, thus the time overhead will be reduced. Another approach is reducing the size of multipart reply message by removing all the non-statistics information. Currently the payload of a single flow's status packet is 88 bytes while in the payload only "Table ID", "Match", "Packet count" and "Byte count" are the information of interest to DECISION, the total size of these fields is only 17 bytes. With the adaptation of multipart message, the traffic of statistics decreases around 80%, thus the transmission time will also be saved.

4.4.2.2 Data Plane Performance in DECISION

In a typical commercial switch, two independent processing units are used to convey the packets in data plane and control plane, i.e., ASIC and CPU. All incoming packets are processed by ASIC unless they are required to be sent to the controller for further analysing. Similarly, all the messages between controller and switch are processed by CPU without the involve-

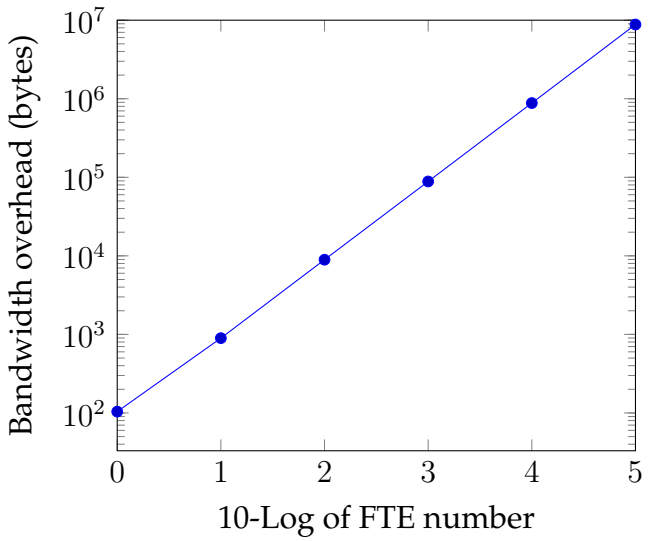


Figure 4.15: Bandwidth overhead of the flow status with increasing number of FTEs (flow match field: MAC address, single flow status length: 88 bytes)

ment of ASIC. In this case, the traffic statistics processing in control plane - which is the place where DECISION overhead occurs - will not affect the packet forwarding performance.

However, in a software switch such as Open vSwitch, only one CPU is used to emulate the behaviour of both control and data planes. In this case, any operations in the control plane will inevitably jeopardise the performance on the data plane.

Figure 4.16 depicts the latency of packet forwarding in a OpenFlow table with 1,000 flows with and without traffic retrieval. As indicated by the x-axis, 50 latencies are collected in each scenario. The test is performed between two vNICs with a fixed packet rate at 39.0 kpps. The traffic is fixed 64-bytes-length UDP packets generated by iperf. In this experiment, the average packet transmission delays with and without statistics retrieval operations are 41.9 μs and 41.26 μs, respectively. It indicates that the latency overhead introduced by traffic statistics processing on Open vSwitch only increases around 1.53%.

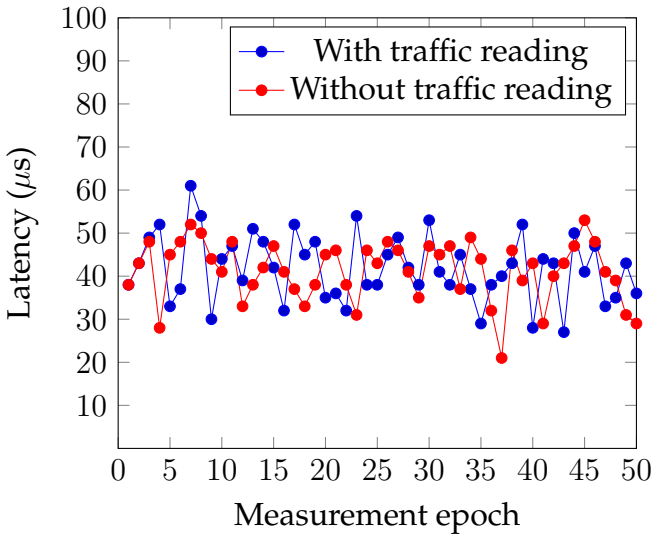


Figure 4.16: Latency comparison

The throughput comparisons under the same circumstances have also been conducted and results are presented in Fig. 4.17. Similar to latency, the throughput is measured with fixed 64-bytes-length UDP packets between two vNICs. The average throughput under no traffic retrieval and consistent traffic data reading are 97.54 kpps and 97.00 kpps, respectively. The actual throughput loss introduced by DECISION's traffic estimation in the control plane is around 0.55%.

Although on Open vSwitch, the packets are forwarded in kernel while the control plane messages are processed in user space, their mutual influences are unavoidable because they rely on the scheduling of a single CPU. However, the experiment has proven that the influence on data plane is comparably less heavy than the traditional FTE installation approach. The direct comparisons between them will be fully exploited in subsection 4.4.2.4.

4.4.2.3 Performance on FTE-installation Based Approach

As illustrated in Fig. 4.10, TM-Tables are reserved for traditional measurements. This can be achieved by installing all TM-FTEs in the monitor-

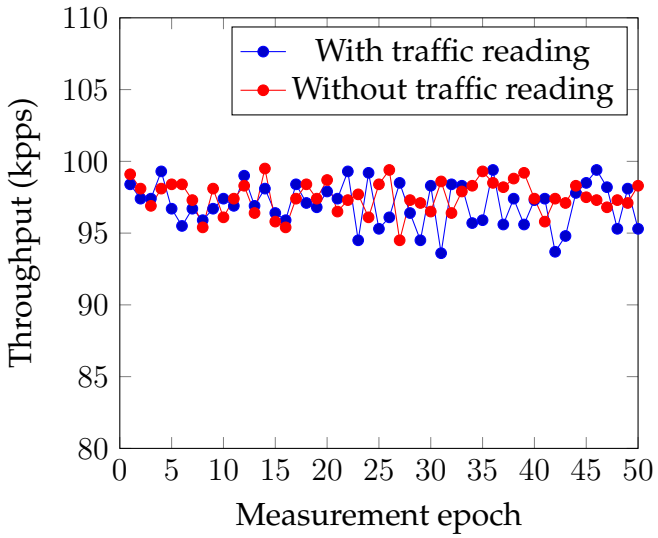


Figure 4.17: DECISION: throughput comparison

ing reserved table(s). However, for multiple parallel measurements, any intersected TM-FTEs must be installed in a different table. Otherwise, a measurement by a TM-FTE with lower priority is not accurate because the incoming packets might hit higher-priority TM-FTEs and be forwarded to other tables.

On Open vSwitch, the single FTE installation time plus the round trip between switch and controller is around $2.1ms$. Since each OpenFlow modification message only contains one flow adding command, the time to install new FTEs is linear to the number of OpenFlow entries. However, every single measurement unusually requires installing one new FTE, the time overhead can be ignored compared to the DECISION computation.

The FTE storage space (measured in number of FTEs) required by the TM-FTE is illustrated on the right-hand of Fig. 4.18, from which it can be observed that the storage space increases up to six orders of magnitude when the measurements of intersected flowsets occur. In the best case, all the match fields of estimated flowsets are independent and these TM-FTEs are installed in a single table; in the worst case, the match fields of every two estimated flowsets intersect and each TM-FTE must occupy a separate

table.

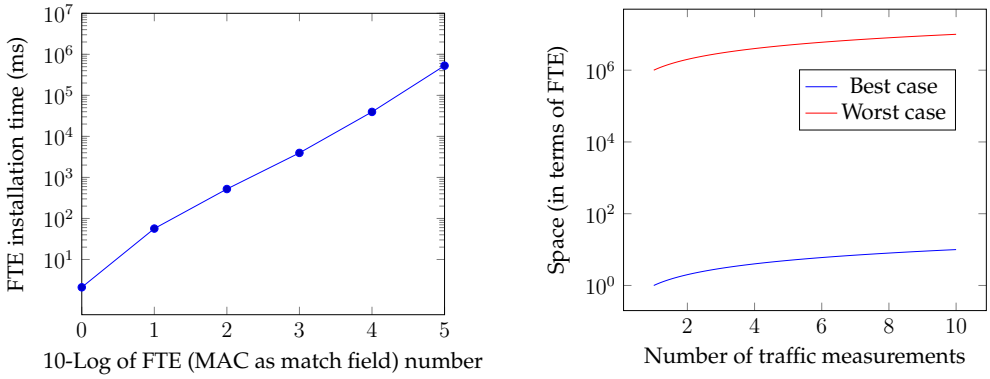


Figure 4.18: TM-FTE installation time and a theoretical estimate of storage space overhead in traditional measurement

4.4.2.4 DECISION vs FTE-installation Approach

Although the direct comparisons between DECISION and FTE-installation approach are not fair for either of them because the major overheads in the former occurs in control plane while the latter takes place in data plane, it is still worth weighing up their overheads before deciding which approach will be taken in the real scenario.

Table 4.5 summarises the performances of the traditional FTE-installation approach and DECISION from four perspectives: control plane, data plane, FTE space and accuracy. Although both time cost and bandwidth overheads in control plane for DECISION and FTE-installation approach are linear to the number of FTEs, the latter greatly outweighs the former in the control plane because usually one new FTE is enough to complete a traffic estimation. However, FTE-installation introduces more overheads in the data plane, i.e., more transmission delay and throughput loss. Actually it is easily demonstrated that DECISION has no any impacts on packet forwarding in physical switch. Another advantage of DECISION is that it relies on exiting FTEs in switch to estimate the traffic, which means no

Table 4.5: Performance comparisons between DECISION and FTE-installation approach

Approaches		DECISION	FTE-installation
Control Plane	Time	Linear to number of FTEs, around six seconds per 10,000 FTE	Linear to number of FTEs (2.1 ms/FTE), one FTE required for each measurement
	Bandwidth	Linear to the number of FTEs, around 8.8 Mbytes per 10,000 flows	Linear to number of FTEs (154 bytes/FTE) one FTE required for each measurement
Data Plane	Delay	On Open vSwitch, 1.53% increase; no additional delay on physical switch	41% increase on physical switch [96]
	Throughput	On Open vSwitch, 0.55% degradation; no additional loss on physical switch	1.81% degradation on physical switch [96]
FTE space		0	best case: one FTE per measurement worst case: one table per measurement
Accuracy		up to 100% (highly dependent on FTE number)	up to 100% (no dependence on FTE number)

OpenFlow table(s) should be reserved in advance.

All the comparisons in Table 4.5 indicate that DECISION is easier to deploy in a production network, but it cannot achieve the same accuracies in most cases. Thus it is more suitable for large-scale networks in which large quantities of FTEs will facilitate DECISION to improve accuracy and the table reservation for FTE-installation approach becomes more unrealistic.

4.5 Summary

Most existing works on networking traffic estimation in SDN rely on the manipulations of the configuration and forwarding policies (rules) in switch [83,85–88]. They perform traffic measurement and estimation by installing application-specified rules and then retrieving their statistics, though their capabilities and scenario vary significantly. As depicted in Table 4.6, DECISION offers the queries for an arbitrary flow set without relying on the manipulation of FTEs. This absolutely non-invasive solution will not alter the existing packet forwarding functionalities or impose any adverse effort on the performance. Although DECISION cannot achieve the same level

accuracy as the traditional FTE installation approach, it outperforms the latter in terms of feasibility and practicality, which makes it more suitable for large-scale networks.

Table 4.6: Summary: DESICION vs existing works

Project	Flow Type	Pipeline Manipulation	Hardware Dependencies	Scenario
ProgME [84]	Flowset	Not Required	N/A	Non-SDN, Heavy Hitter Detection
iSTAMP [86]	Aggregated	FTE	TCAM	SDN, Generic Purpose
DCM [87]	Aggregated	FTE	TCAM/FPGA	SDN , Generic Purpose
OpenSketch [83] [85]	Flow	Sketch	TCAM/FPGA	SDN , Generic Purpose
OpenWatch [88]	Flow	FTE	Not Mentioned	SDN, Anomaly Detection
DECISION	Flowset	Not Required	N/A	SDN, Generic Purpose

In the next chapter, the last research contribution will be discussed. It is a traffic anomaly detection based on pattern recognition and FTE correlation. The research scope has been further extended to multiple switches and their associated traffic statistics.

Chapter 5

Heavy Hitter Detection & Identification

In a large network, it is often important to be able to detect high-volume traffic in near real-time. Existing work on the detection and identification of such high volume traffic (so-called *heavy hitters*) is typically delegated to individual nodes and often relies on deep packet inspection or packet sampling. However, these techniques have well known limitations in terms of its ability to scale with network size. Inspired by the capabilities of SDN, a novel heavy hitter detection solution based on understanding connections between traffic statistics and OpenFlow rules has been proposed and verified in this chapter. This approach relies on mining traffic statistics (e.g. port bitrate) and forwarding table entry to improve heavy hitter detection. The rationale behind this approach are (i) the information is readily available with minimal overheads, thus it scales better with increasing network size; and (ii) the FTEs and traffic statistics provide different vantage for detection and identification of heavy hitters. The effectiveness and accuracy of the proposed heavy hitter detection algorithm have been implemented and evaluated on a test bed as a proof-of-concept. The test results show that this heavy hitter detection simultaneously achieves considerable accuracy and good scalability.

5.1 Research Question

The phenomena of small number of flows carrying majority of bytes has been studied in many incarnations in computer networking [99, 100]. A simple way to define these "important" traffic are the terms *elephant flow* or *heavy hitter* (HH). They can be identified from the fields in a packet header, for example, the source IP prefix or destination MAC address. The detection and identification of these flows are critical for Quality of Service provisioning and traffic load balancing.

To find these heavy hitters in a network, two general approaches have been widely used. One is based on statistical sampling and another utilises a specialised data structures called a "sketch" to maintain a probabilistic record of data streams.

There are two major problems with the existing approaches. The first problem is excessive overheads. Sampling-based technique is extremely resource intensive because of the packet inspection process while sketch-based solution is highly dependent on proprietary hardware. Another problem is that they lack the capability to detect the multi-dimensional heavy hitters (MHH) because the packet header fields (e.g., MAC address, IP address and port numbers) must be defined in advance for accurate detection. Here the "dimension" refers to the type of fields to identify a flow, for example, all the packets with the same destination IP address. If there is more than one field to identify a heavy hitter flow, this heavy hitter is also called multi-dimensional heavy hitters [101]. However, it is difficult to identify which header fields to monitor because MHHs are aggregate flows that have headers permuted from its constituent flows. These have been shown on the traffic originated from virtual machines (VMs) in which the traffic is aggregated from various applications. Without prior knowledge about the patterns of such traffic, all possible combinations of aggregates must be examined to detect the heavy hitters.

In the study of multi-dimensional traffic, one notable type of heavy

hitter is called hierarchical heavy hitter (HHH) [102, 103]. Autofocus is a good attempt to automatically characterise HHH traffic based on IP addresses [101]. Autofocus is a method of traffic characterisation that automatically groups traffic into minimal clusters of conspicuous consumption. Rather than providing a static analysis specialized to capture flows, applications, or network-to-network traffic matrices, this approach dynamically produces hybrid traffic definitions that match the underlying usage. However, the research on unstructured heavy hitters has not been fully explored. There are numerous examples that show the aggregate traffic does not follow any known pattern [104].

To address these problems, a lightweight heavy hitter detection and identification solution has been proposed and implemented which incurs lower communication and processing overheads compared to traditional sampling-based technology. It is a software solution based on multiple rounds of retrieving and analysing network traffic statistics. A controller can retrieve statistics from the switches at three levels: (i) OpenFlow table, (ii) aggregate flow, and (iii) individual flow. By analysing these coarse-to-fine statistical information, the suspected HHs are gradually narrowed down and finally identified. It has been demonstrated on a test bed that the proposed solution meets the functional tests for HH detection and it is also shown via analysis that this solution is scalable and efficient.

5.2 Heavy Hitter Detection

5.2.1 Design of the HH Detection Framework

A generic SDN application framework that retrieves and processes the network's information in a hierarchical and staged manner is proposed to reduce the: (i) computation overheads on the controller and (ii) communication overheads between switches and controller. In comparison to a sFlow or NetFlow server in a conventional network, a centralised control-

ler in SDN has a broader view of the devices under its control. An SDN controller can leverage real-time status and statistics from these devices to make better decisions about how to deploy or optimise them for HH detection. However, it is infeasible to request every switch to report all flow statistics due to the enormous communication overheads in a network and the excessive processing to be done by the controller.

Besides the switch-level and table-level (coarse-grained) statistics, a controller also deals with more fine-grained statistics such as the status of ports, meter, CPU queue, etc. [2]. The key challenge of designing an SDN-based HH detection solution is to strike a careful balance between generality (supporting a wide variety of statistics) and efficiency (low communication cost and computation overheads) [83].

Two types of inputs are used in the HH detection framework: traffic statistics and OpenFlow tables. Traffic statistics enable the controller to understand the network runtime status while OpenFlow tables define the underlying paths across the network. To avoid input deluge to the controller, a multi-stage coarse to fine approach is used to stagger the inputs for HH detection. A controller begins with the coarse-grained information such as table or port level statistics for the initial investigation, and then selects the fine-grained flow-level information for further processing. This two-stage input processing design allows the controller to focus on a small set of information in each stage, thus reducing communication and computation costs.

The rationale behind the multi-stage approach is that aggregate traffic retains some characteristics of each individual flow, especially the most significant one. In fact, all traffic can be considered as aggregate flows with different granularities. The traffic in a single Ethernet card is an aggregation of flows from all applications on that server; the traffic in a physical switch port is an aggregation of flows from several servers or switches. Early traffic models were derived from telecommunications networks and generally operated under the assumption that aggregate traffic from a

large number of sources smooths out bursts [105]. However, today's networks are designed to scale-out rather than to scale-up, which makes these aggregate traffic traceable and their hidden HHs detectable [106].

5.2.2 Modules of the HH Detection Framework

The HH detection framework consists of a detection module and an identification module. The detection module determines if an HH is likely to be present in a flow while the identification module positively establishes the HH. The worklow for the HH detection and identification framework in Fig. 5.1 depicts two-stages and each stage is instantiated as a module.

In Stage 1 (detection module) of Fig. 5.1, the SDN controller requests all switches to report their coarse level information such as OpenFlow table status or port packet counters. After examining the coarse level information, the controller filters out the messages in which it has no interest. Then the controller requests switches to send the finer aggregate statistics (e.g. per-flow level) for further processing. Since FTEs determine the forwarding behavior of all the underlying flows, the characteristics of these flows can be reversely inferred from its matched FTEs.

In Stage 2 (identification module) of Fig. 5.1, a new FTE in which the match fields are constructed based on the characteristics of the detected heavy hitter is created and installed into the switches along the path where HH are detected. By monitoring the traffic statistics of these new added FTEs, it can be verified whether a detected flow is indeed an HH or just an aggregation of multiple mice flows.

5.2.3 Detection Module

The purpose of detection module is to distinguish the elephant flows from each traffic data stream and group them based on certain measure of similarity. This is achieved through processing the information from switches such as outgoing counters or bytes as a time series. The time series model

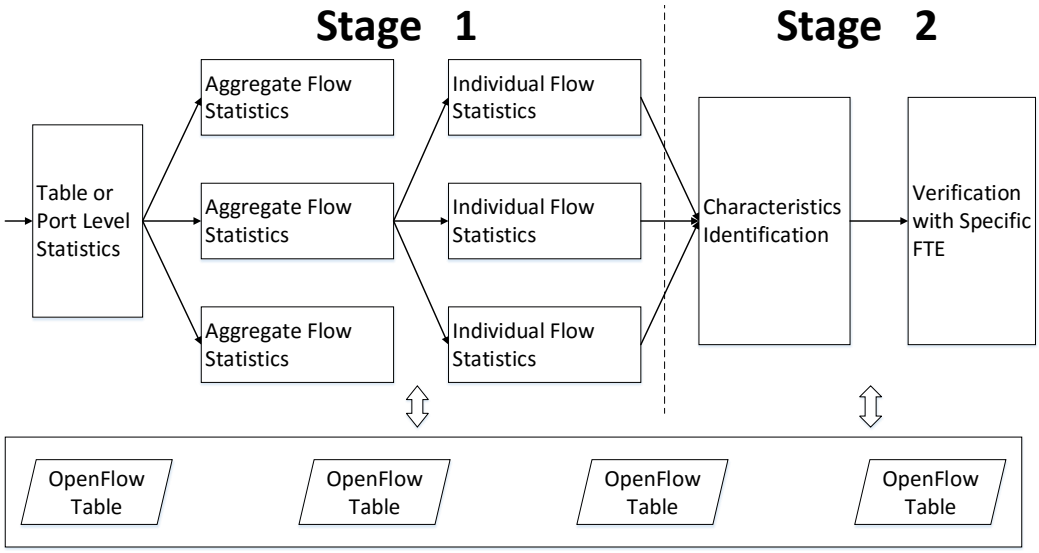


Figure 5.1: The proposed HH detection framework as a two-stage process. Each stage is instantiated as a module.

[107] is adopted to describe the traffic from a physical port in a switch. It is a sequence of traffic data points which consist of successive measurements made over a constant time interval.

As illustrated in Fig. 5.2, the detection module consists of four procedures: smoothing, thresholding, windowing, and correlating. Smoothing removes the spikes in the time series, thresholding determines the interval in which an elephant flow occurs, windowing isolates these elephant flows after applying a threshold. The effect of these three procedures on a raw traffic stream is illustrated in Fig. 5.9. Finally, these windowed datasets are grouped based on their correlations and each group represents a set of elephant flows with high similarities, assumed to be the same elephant flow pending further diagnostic. A heavy hitter is defined as follows:

Definition 5.2.1 (Heavy Hitter)**.** $\mathscr{L}$ is a set of points indexed by k for a single flow such that $\mathscr{L} = \{\alpha_k\} : \forall k \in \mathbb{Z}_0^+$. Let L denote the minimum period for observing an HH, and let I denote maximum period in which all sampled rates are less than the threshold T; an HH $\mathscr{H}$ is a subset of $\mathscr{L}$, $\mathscr{H} = \{\alpha_k\}_{k=i}^{j} : (j-i+1) \geq L, (\nexists m, n : i \leq m \leq n \leq j, (n-m+1) \geq$

$I, \{\alpha_k\}_{k=m}^{n} < T)$.

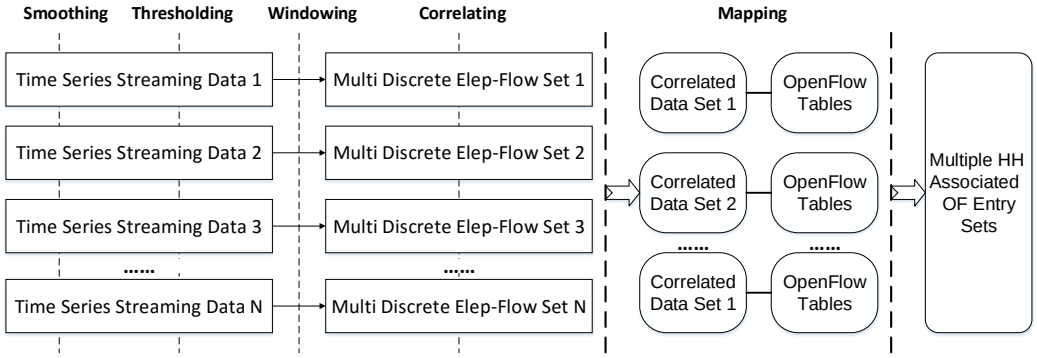


Figure 5.2: The work-flow of four procedures within the detection module.

5.2.3.1 Smoothing

Smoothing is the procedure that removes "noise" from a data stream. For an aggregate traffic dataset, the elephant flows are considered as the desired signal and the mice flows as noise. We use the k-nearest neighbour algorithm to filter out spikes and noise from the time-series of an aggregate flow and this procedure is outlined in Algorithm 10.

5.2.3.2 Thresholding

After smoothing the incoming traffic dataset S_i (where i is the index of that dataset), a threshold is applied to S_i. All the flows above a certain threshold value T are considered as potential elephant flows. The thresholding process is expressed in the following equation:

$$T_i = \begin{cases} S_i - T, & if \quad S_i \geq T; \\ 0, & if \quad S_i < T. \end{cases} \tag{5.1}$$

Algorithm 10 Minimum k-nearest neighbours algorithm

```
1: procedure SMOOTHING(k)
2:     R ← input: Two dimensional array of raw traffic data
3:     S ← output: Two dimensional array of smoothing data
4:     M ← Absolute maximum value
5:     for i = 0, i < len(R)/k, i++ do
6:         for j = 0, j < k, j++ do
7:             if R_{i*k+j} < M then
8:                 M ← R_{i*k+j}
9:             end if
10:        end for
11:        S_i ← M
12:    end for
13: end procedure
```

5.2.3.3 Windowing

In practice, T_i cannot be observed on the interval $(-\infty, +\infty)$. One way of overcoming this problem is to split every traffic stream into multiple fixed time interval datasets. According to Definition 5.2.1, an elephant flow or HH is a flow in which the longest continuous period below the given threshold is no more than t with duration no less than l. For the post-threshold dataset T_i, the problem of identifying potential HH is simplified to finding all datasets with the following three attributes: i) the length of each dataset is no less than l; ii) the number of consecutive zeros is smaller than t; and iii) begin and end with non-zero value. This windowing procedure is shown in Algorithm 11.

5.2.3.4 Correlating

To reduce the dimensionality of the original data, a series of similarity search methods with a multi-dimensional index structure to index the data in the transformed space has been proposed [108]. Different representations determine the ease and efficiency of similarity identification. Four indexable feature extraction techniques are simulated and verified with

Algorithm 11 Windowing

```
1: procedure WINDOWING
2:     T ← input: Two dimensional array thresholding data
3:     W ← output: Two dimensional array windowing data
4:     Z ← Array to save consecutive 0 in T
5:     t ← Treat as same flow with consecutive 0 less than t
6:     l ← The minimum length of a window
7:     n ← 0, Number of windows
8:     for i = 0, i < len(T), i++ do
9:         if T_i = 0 then
10:            if (len(W_n) ≥ l) & (len(Z)) ≥ t) then
11:                n ← n+1
12:            end if
13:            Append T_i to Z
14:        else
15:            if T_i-1 = 0 &
                  (len(W_n) > 0) & (len(Z) < t) then
16:                Append Z to W_n
17:            end if
18:            Z ← 0
19:            Append T_i to W_n
20:        end if
21:    end for
22: end procedure
```

the windowed data: Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), Piecewise Linear Approximation (PLA) and Adaptive Piecewise Constant Approximation (APCA) [109].

An original signal is transformed into the frequency domain (DFT) or decomposed in terms of a basis set of functions (DWT). Each time series can be represented by a few selected coefficients corresponding to the low frequencies in DFT or scaling functions in DWT. These few coefficients preserve most of the energy of the original signal. The particular wavelet chosen in comparison is the simplest wavelet form namely the Haar Wavelet. PLA projects a curve into a series of segmented straight-line. In the implementation, the least-square method (LSQ) is used to obtain an

optimal line to fit the non-linear curve. Different from the aforementioned three representation techniques, APCA approximates each time series as a set of constant value segments of *varying* lengths.

Taking the windowed dataset of sub-plot "Traffic Data Windowing-2" that appear earlier in Fig. 5.9 as an example, the above mentioned four representations is illustrated in Fig. 5.3. From left to right, the four columns demonstrate the signal transformation with algorithm DFT, DWT, PLA and APCA, respectively. The first row is the original signal, and the rest three rows are the representation with resampling rate at 1/64, 1/16 and 1/4, respectively. In the test scenario, the size of the original dataset is 256, thus the number of resampling points from row two to row four in Fig. 5.3 are 4, 16 and 64, respectively. It can be observed that APCA outperforms the other three algorithms at the resampling rate 1/64 and 1/16 while all the feature selections except PLA reflect the main aspects of the original signal at rate 1/4. The varying length of each dimension in APCA fits well with elephant flows which usually last for a certain time in a relative stable volume, and lets APCA reflect the key features of traffic data with less storage space.

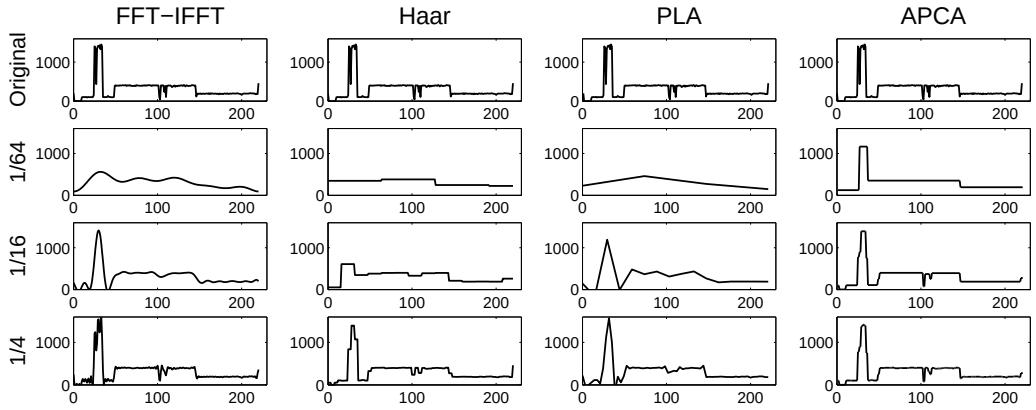


Figure 5.3: The most common representations for time series data mining: DFT, DWT(Haar Wavelet), PLA, APCA

If an HH traverses multiple switches, it can be observed from the stat-

istics of the FTEs in these switches. The procedure of *correlating* is used to find the HHs belonging to the same flow based on their similarities. There are two major categories of similarities: bias oriented and variance oriented. If bias outweighs variance as an indicator of similarity, the bias oriented similarity is selected. In this case, two datasets with exact fluctuations will be considered as dissimilar if the difference of their mean values exceed a certain threshold.

On the contrary, the variance oriented similarity values the weights of deviations more than the difference of their mean values. For an aggregate traffic stream, the variance of an elephant flow is not easily identified because it is obscured by other mice flows. In this solution, the bias oriented similarity will be applied to find the possibility whether two aggregate flows hide the same elephant flow.

The similarity between two time series is typically measured with Euclidean distance. The Euclidean distance is suitable for HH detection purpose because there are no offset and acceleration-deceleration along the time dimension to be taken into consideration after the time series passes through the windowing procedure. However, the problem of storing large amounts of data persists because of the following three factors prevalent in large networks: (i) over hundreds of time series data streams; (ii) large volume per dataset (since each unit of sampling traffic data takes one byte in memory, the total size of a single 60-minute dataset will be as high as $3.6MB$); and (iii) fast response time for detecting HH.

In this chapter, Adaptive Piecewise Constant Approximation (APCA) [109] is adopted to represent the original time series data for similarity search. The APCA representation of a time series $C = \{c_1, \ldots, c_n\}$ can be expressed as:

$$C = \{\langle cv_1, cr_1\rangle, \ldots, \langle cv_M, cr_M\rangle\}, cr_0 = 0 \tag{5.2}$$

where cv_i is the mean value of datapoints in the *i*-th segment and cr_i is the right endpoint of the *i*-th segment.

5.2.4 Detecting Similarity

The approximate Euclidean distance between a measured dataset and a time series in APCA representation was derived in [109]. Considering the APCA representation C as a reconstructed time series, the Euclidean distance between a time series Q and C is defined as:

$$D_{AE}(Q,C) = \sqrt{\sum_{i=1}^{M} \sum_{i=1}^{cr_1 - cr_{i=1}} (cv_i - q_{k+cr_{i=1}})^2}. \tag{5.3}$$

To further reduce the computation time, a coarser approximation of Euclidean distance between two time series $A = \{\langle av_1, ar_1\rangle, \ldots, \langle av_m, cr_m\rangle\}$ and $B = \{\langle bv_1, br_1\rangle, \ldots, \langle bv_n, br_n\rangle\}$, $D_{CE}(A,B)$, is used in this solution. This distance is calculated directly based on their respective APCA representation and defined as:

$$D_{CE}(A,B) = \sqrt{\sum_{i=1}^{m} \sum_{i=1}^{n} (av_i - bv_j)^2 |ar_i - br_j|}, \tag{5.4}$$

where $(ar_{i\text{-}1} \leq br_j \leq ar_i)$ or $(br_{j\text{-}1} \leq ar_i \leq br_j)$.

Algorithm 12 Euclidean distance of APCA representation

```
1: procedure APCA_DISTANCE
2:     A, B ← input: Two APCA datasets
3:     E, AR, BR ← 0 // temporary vectors
4:     AV ← all av_i in A;    BV ← all bv_i in B
5:     D ← output: Euclidean distance of A and B
6:     for ar_i in A do
7:         Append ar_i to E and AR
8:     end for
9:     for br_i in B do
10:        Append br_i to E and BR
11:    end for
12:    Sort, Insert 0 at E[0] and delete E[len(E) − 1]
13:    a_value ← AV[0];    b_value ← BV[0]
14:    for i = 0, i < len(E) − 1, i++ do
15:        if E_i in AR then
16:            index ← the index of item E_i in AR
17:            a_value ← AV_index
18:        else
19:            if E_i in BR then
20:                index ← the index of item E_i in BR
21:                b_value ← BV_index
22:            end if
23:        end if
24:        D+=(|a_value − b_value|) ∗ (E_{i+1} − E_i)
25:    end for
26: end procedure
```

Although the distance $D_{CE}(A, B)$ is not a lower bound measure, it is still very useful for approximate search in a large dataset. The procedure to compute the distance between APCA representations is described in Algorithm 12.

Fig. 5.10 illustrates the APCA distance computation of two datasets, each of them consists of 16 elements.

APCA distance will be used to determine the similarity of two HHs. If their distance is below a given threshold which is also called similarity

threshold (ST) in this chapter, they will be considered as the same flow.

5.2.5 HH Detection via Match Set Analysis

A typical OpenFlow FTE is composed of *match set*, *actions(instructions)* and *priority*, this is shown in Fig. 5.4. Usually, a match set is composed of multiple fields which identify an individual flow or a set of flows. In the medium access control (MAC) layer, it includes destination MAC (DMAC) and source MAC (SMAC) addresses, while in the network layer, it is a 5-tuple consisting of source IP address (SIP), port number, destination IP address (DIP), port number and the protocol in use. In most cases, actions contain the output ports, which helps relate traffic statistics and their corresponding FTEs. Priority indicates matching precedence of a flow entry which means the matching entry with highest priority will determine how a packet is forwarded.

Match	Priority	Counters	Instructions	Timeouts	...

Figure 5.4: A single flow table entry

The relations that can be in place among match sets and instruction sets were first analyzed in [3]. Based on the potential relation combinations, they define five FTE operator types: *Disjoint*, *Exactly matching*, *Subset*, *Superset*, and *Correlated*. A brief overview of these operators is given in Appendix B.

An HH that matches multiple FTEs, can be expressed by a function of match fields and set operators (explained in Appendix B). This approach of using one or more match fields to identify an HH is also called *HH match set*. Suppose there are two OpenFlow switches S_A and S_B, each of them maintains a single OpenFlow table, which are shown in Table 5.1 and Table 5.2, respectively. A controller detects two elephant flows coming from port 21 in switch S_A and port 4 in switch S_B, respectively. These two flows demonstrate a certain level of similarity such that they are assumed

as a same flow. If so, this elephant flow will match at least one FTE in both tables simultaneously.

For Table 5.1, the potential FTE matching set is $F_A = \{2, i+1, i+2\}$ because all outputs of these FTEs consist of port 21 where elephant flow is observed. Similarly for Table 5.2, the FTE set is $F_B = \{1, 3, j+1, j+2\}$. According to the definition of conjunction and disjunction, the match set of packets that matches at least one FTE in both F_A and F_B can be expressed as below:

$$\begin{aligned} M_{F_{A\wedge B}} =& M_{F_A} \wedge M_{F_B} \\ =& \big(M_2^A \vee M_{i+1}^A \vee M_{i+2}^A\big) \wedge \big(M_1^B \vee M_3^B \vee M_{j+1}^B \vee M_{j+2}^B\big), \end{aligned}$$

such that $M_{F_{A\wedge B}}$ consists of one or more match fields which determines the characteristics of matching packets, i.e., it is an HH match set.

Table 5.1: OpenFlow sample table A

Index	FTE
1	Match: M_1^A, Action: Drop
2	Match: M_2^A, Action: output 21
3	Match: M_3^A, Action: output 2
i	...
$i+1$	Match: M_{i+1}^A, Action: output 2,21,24
$i+2$	Match: M_{i+2}^A, Action: output: ALL
$i+3$	Match: ALL, Action: output: CONTROLLER

Table 5.2: OpenFlow sample table B

Index	FTE
1	Match: M_1^B, Action: output 4
2	Match: M_2^B, Action: output Drop
3	Match: M_3^B, Action: output 4, 18,22
j	...
$j+1$	Match: M_{j+1}^B, Action: output 4,20
$j+2$	Match: M_{j+2}^B, Action: output: ANY
$j+3$	Match: ALL, Action: output: CONTROLLER

5.3 Heavy Hitter Identification

Once a suspected HH is found, the next step is to verify whether it is a real HH or a false positive. The approach is to install a newly constructed FTE with the match fields as the HH match set into the switches where the HH occurs and then observe its statistics. One challenge is that the behaviour of the existing FTEs should not be affected by the new added

flow entry. However, the newly added FTE might share the same match fields with the existing FTEs which also match the suspected HHs. The HH will not match with the original FTE any more if the new FTE is installed in the same table with higher priority. Thus, these FTEs must be carefully constructed to ensure the existing pipelines of all packets remain unchanged.

Using the "multi-table" feature provided by the latest OpenFlow specification [2], these new FTEs will be added in a separate table, table "0", to guarantee that all packets match against the FTEs in this table while preserving the original actions being executed. Table "0" is specifically designed for HH verification. The process is illustrated in Fig. 5.5 in which all the instructions of the FTEs in Table 0 have been set as "Goto Table: 1". In Table 0, all packets will be directed into next table for normal processing without performing any additional actions. The counters of these new FTEs in Table 0 are recorded in switch side. The controller retrieves these statistics of these FTEs to validate whether an HH match set really represents an elephant flow.

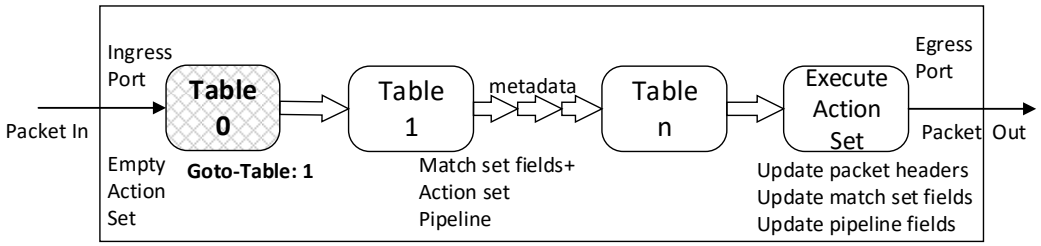


Figure 5.5: Table "0" reservation for HH identification in OpenFlow

5.4 Experiment Evaluation

5.4.1 Test Bed & Traffic

A simple leaf-spine topology is adopted in the experiment which uses one Ryu OpenFlow 1.4.0 controller, three physical switches and eight hosts. Fig. 5.6 illustrates the topology of the data plane in which the spine switch ("Pica8-2") connects the other two leaf switches, "Pica8-1" and "Pica8-3", respectively. The spine switch forwards packets according to the destination MAC as match set, while leaf switches forward packets according to the destination IP address as match set. All hosts are in the same IP subnet and all FTEs are installed proactively in the switches to ensure connectivity between hosts. More details of the test bed and the source code of this project are listed in Appendix D.

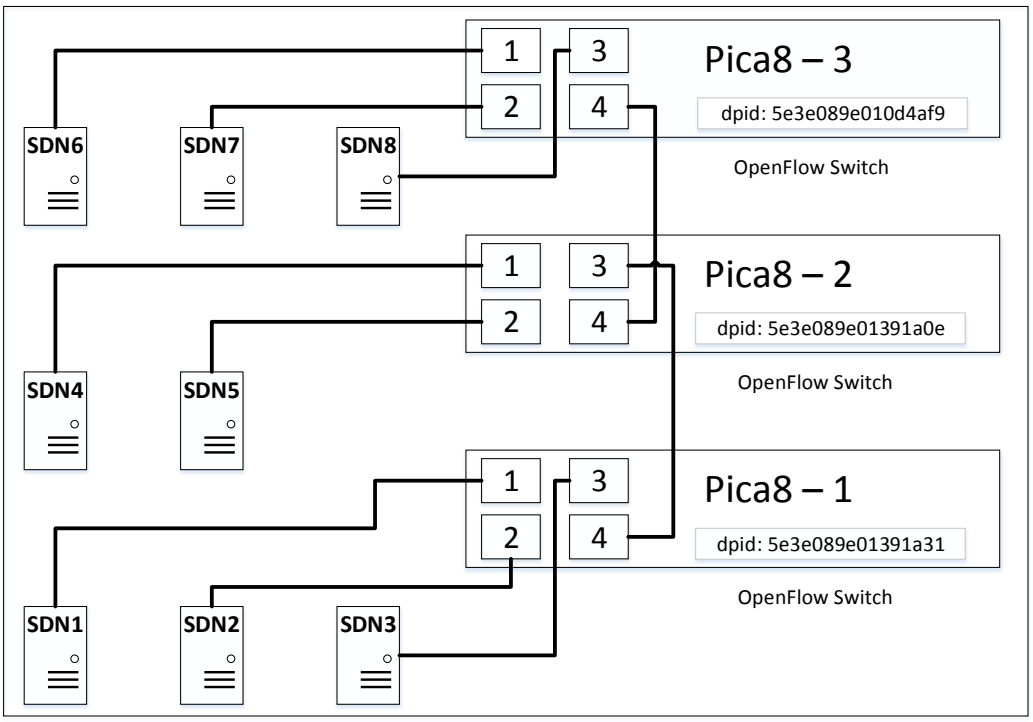


Figure 5.6: Test bed data plane topology

To demonstrate the functionality of the HH detection solution, Distributed Internet Traffic Generator (D-ITG) [110] is chosen for traffic generation as it supports the following customization: i) number of flows per physical port; ii) transmission rate of each flow; iii) variance of the transmission rate for each flow; and iv) protocol, duration, jitter, and delay of each flow. D-ITG produces packet level traffic with customizable packet interarrival time and packet size.

A *distance-based HH detection system* (DHHDS) is implemented in Python with three independent threads for traffic statistics retrieval, data logging and HH detection, respectively. The traffic rate threshold T (see Eq.(5.1)) is adjusted to the mean traffic rate of all monitored flows. Likewise, the similarity threshold ST is also tuned to the average APCA distance. Thus, any two or more flows will be considered as a single HH if their traffic rates and APCA distance are below the average transmission rate and distance respectively.

The traffic data streams retrieved from all active physical ports are continuous in the time domain. In this evaluation, each stream is split into a series of discrete datasets with fixed time intervals. Recent research shows that elephant flows maintain their state for 20-40 minutes [111]. In this chapter, the length of each dataset is set as 60 minutes to guarantee most of elephant flows be covered by two consecutive datasets.

5.4.2 Functional Test

A functional test checks the detection correctness by comparing the results for a given input traffic set against the desired outcome. In the functional test, seven sets of test traffic have been defined, each of them has various number of flows and HHs.

The process of detection in the functional test is shown in Fig. 5.7. The controller retrieves the real-time outgoing traffic data of all physical ports and then analyse these data and their associated FTEs to determine

whether there are some HHs. Suppose there is a time period in which one flow egressing out of port 2 in switch A and another flow coming from port 1 in switch B exceed the average transmission rate simultaneously, which implies that both flows might contain HHs. If their APCA distance is also less than the average distance of all potential HHs, they will be assumed to be the same HH. This HH matches all the flow entries associated with these two ports. As illustrated in Fig. 5.7, the match set of output with port 2 in switch A and port 1 in switch B are "DMAC: 00:11:22:33:44:55" and "DIP: 192.137.1.1/32" respectively. Thus it can be determined that this HH goes to a machine with MAC address as 00:11:22:33:44:55 and IP address as 192.137.1.1 by the conjunction operation of these two match sets. In the identification stage, a new FTE with the HH match set {*"DMAC: 00:11:22:33:44:55", "DIP: 192.137.1.1/32"*} as the match fields will be installed and monitored.

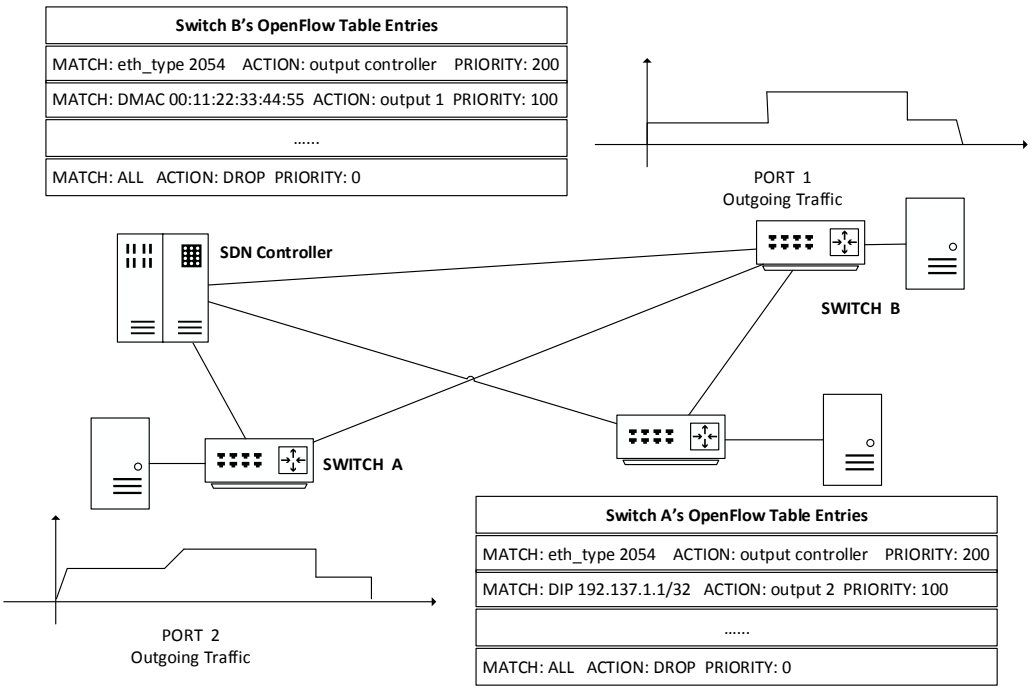


Figure 5.7: A heavy hitter detection scenario

Table 5.3: Heavy hitter functional test results

Set	Num of flows	**Num of HH**	**HH Traffic percentage**	**Packet size distribution**	**Accuracy**
1	3	1	81%	Constant	100%
2	10	2	$42\%, 42\%$	Constant	50%
3	10	2	$42\%, 32\%$	Constant	75%
4	10	2	$42\%, 42\%$	Uniform $[8KB, 50KB]$	50%
5	10	2	$42\%, 32\%$	Uniform $[8KB, 50KB]$	75%
6	10	2	$42\%, 42\%$	Poisson $(61KB)$	50%
7	10	2	$42\%, 32\%$	Poisson $(61KB)$	75%

Table 5.3 shows the functional test results of DHHDS with seven different sets of input traffic. As shown in Table 5.3, these flows follow certain packet size distributions and contain different percentages of HHs. The detection accuracy, also called positive predictive value (PPV), is determined by the true positive (TP) and false positive (FP), which is expressed as:

$$PPV = TP/(TP + FP).$$

True positive means that a flow is correctly identified as HH while false positive incorrectly labels a flow as an HH. The results show that the detection accuracy is dependent on each flow's transmission rate. If two different flows demonstrate a certain similarity in a time period, DHHDS might consider them as the same flow incorrectly. This explains the 50% detection accuracy in some cases because DHHDS cannot distinguish any two HHs with the same transmission rate. The scenario whereby two flows share the exact transmission rate is rare. Further improvements to the detection accuracy can be achieved by tuning the similarity threshold.

Another factor related to accuracy is the misidentification of an aggregate elephant flow as an HH. It happens when multiple flows share a common forwarding path between two intermediate switches, which can be avoided if at least one port in edge switches is involved in the similarity computation. Another observation is that DHHDS is not sensitive to the traffic pattern. This is attributed to the APCA representation, which aver-

ages time series traffic and thus the distance between two APCA datasets are comparably insensitive to short traffic spikes.

The tests demonstrate that the HH detection using traffic statistics and FTE functions is as expected. However, its efficiency needs to be evaluated to determine whether it can be deployed in a large-scale network.

5.5 Performance Analysis

In the functional test of DHHDS, a real test bed with physical switches and servers is used. However, the bottleneck in DHHDS is the communication overhead and computation capability of the controller. This is because the controller collects and analyses statistics of all physical ports, which might affect the network performance as the network scales up. In this section, the time overheads in a network with fixed number of ports is analysed and then the number of ports are repeatedly increased to evaluate the scaling capability of DHHDS.

5.5.1 Evaluation Scenario

Since both communication overhead and computation time are highly dependent on the traffic statistics which are measured on the level of physical ports, the number of active ports will be the evaluation units. Scaling to various number of switches and ports in the laboratory-level test bed is prohibitively expensive, therefore this scenario is simulated by replaying scaled up traffic statistics from a test bed (recall that DHHDS relies on traffic statistics from switches). In this performance evaluation, the controller is a desktop machine (Dell Optiplex 9020) with Intel Core i7-4790 CPU at 3.60GHz and 8Gb RAM running the Ryu controller. Besides, it also assumed that all switches' port are operating at 10Gbps and 10% of these ports contain heavy hitters.

5.5.2 HH Detection Time

The time to complete a round of heavy hitter detection in DHHDS (T_{HH}) can be expressed as:

$$T_{HH} = T_{comm} + T_{switch} + T_{controller},$$

where T_{comm} represents the communication time between controller and switches, T_{switch} denotes the querying time for traffic statistics and $T_{controller}$ denotes the time to process traffic statistics at the controller side. Recent measurement studies have shown that both T_{comm} and T_{switch} are in order of milliseconds [112, 113] compared to seconds for $T_{controller}$, and can be safely ignored in computing measures of efficiency.

The processing time at the controller side, $T_{controller}$, is proportional to the number of active ports. In the scenario of a network with 4800–ports (100 switches$\times$48 ports), the computation time is 15.2 seconds. As shown in Fig. 5.8, when the number of active ports are increased from 10^2 to 10^5, the HH detection time increases linearly from less than one second to around five minutes. The five minute benchmark is the average time for HH detection for sampling-based methods [114] and this is shown as a horizontal line in Fig. 5.8.

Upon profiling the operations and computations in DHHDS, it is found that the most time-consuming procedures is APCA which costs more than 98% of the total computation time. This is also consistent with the fact that the complexity of APCA is $O(n\log(n))$ while the remaining procedures (smoothing, thresholding and windowing) are $O(n)$, where n denotes the length of sampled traffic data.

5.5.3 Comparing Different HH Approaches

Table 5.4 summarises the major differences and indicative performances for three widely documented HH detection approaches in the 4800–port

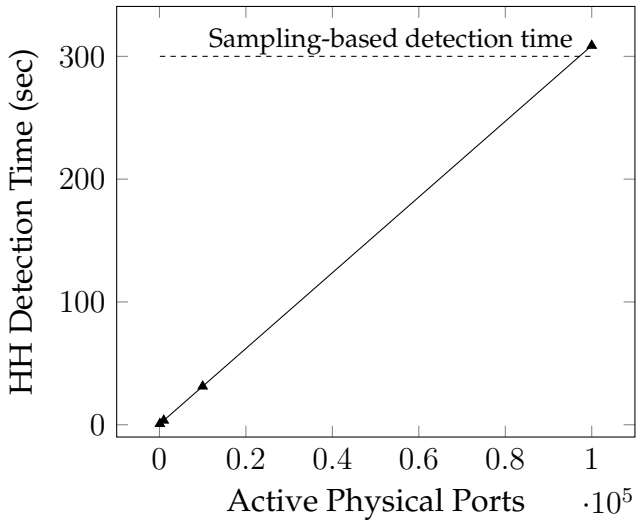


Figure 5.8: HH detection time with increasing number of active ports.

scenario. In this table, the values for computation time in the sampling based approach is calculated with the assumption that the link utilization rate is 20% and the sampling rate is 5% given by [115].

Table 5.4: Performance comparisons of HH detection approaches

	Objects Measured	Scale	Scanning(Computation) Time	Memory	Communication overhead
Sampling based	Packets	Multiple devices	∼510s (DFA OP [115])	7391KB (DFA OP [115])	Order of Gbps
Sketch based	Hash Table	Device-level	Line rate [83]	∼2MB [83]	Order of Kbps
Proposed DHHDS	Statistics	Multiple devices	∼15s	4915KB	Order of Mbps

Comparing the communication overheads, it is clear that sampling-based solution is of limited use in large-scale networks. Sketch-based solutions have low communication overheads because they are self contained in individual switches, but it requires hardware customization making it less favourable with network operators. The proposed solution manages the traffic statistics rather than the packets to achieve a balance between the resource consumption and efficiency.

5.6 Summary

Motivated by SDN, this chapter tackles the problem of heavy hitter detection and identification without packet inspection. The proposed solution relies on analysis of the readily available information in the controller and is especially suitable for large-scale networks. Compared with traditional HH detection, this solution has three advantages: i) easily scalable; ii) arbitrary traffic detection without prior knowledge of traffic patterns; and iii) independent of underlying hardware design.

In the next chapter, all the afore-mentioned research contributions will be summarised. Their implications in real world network and potential future research points will also be discussed.

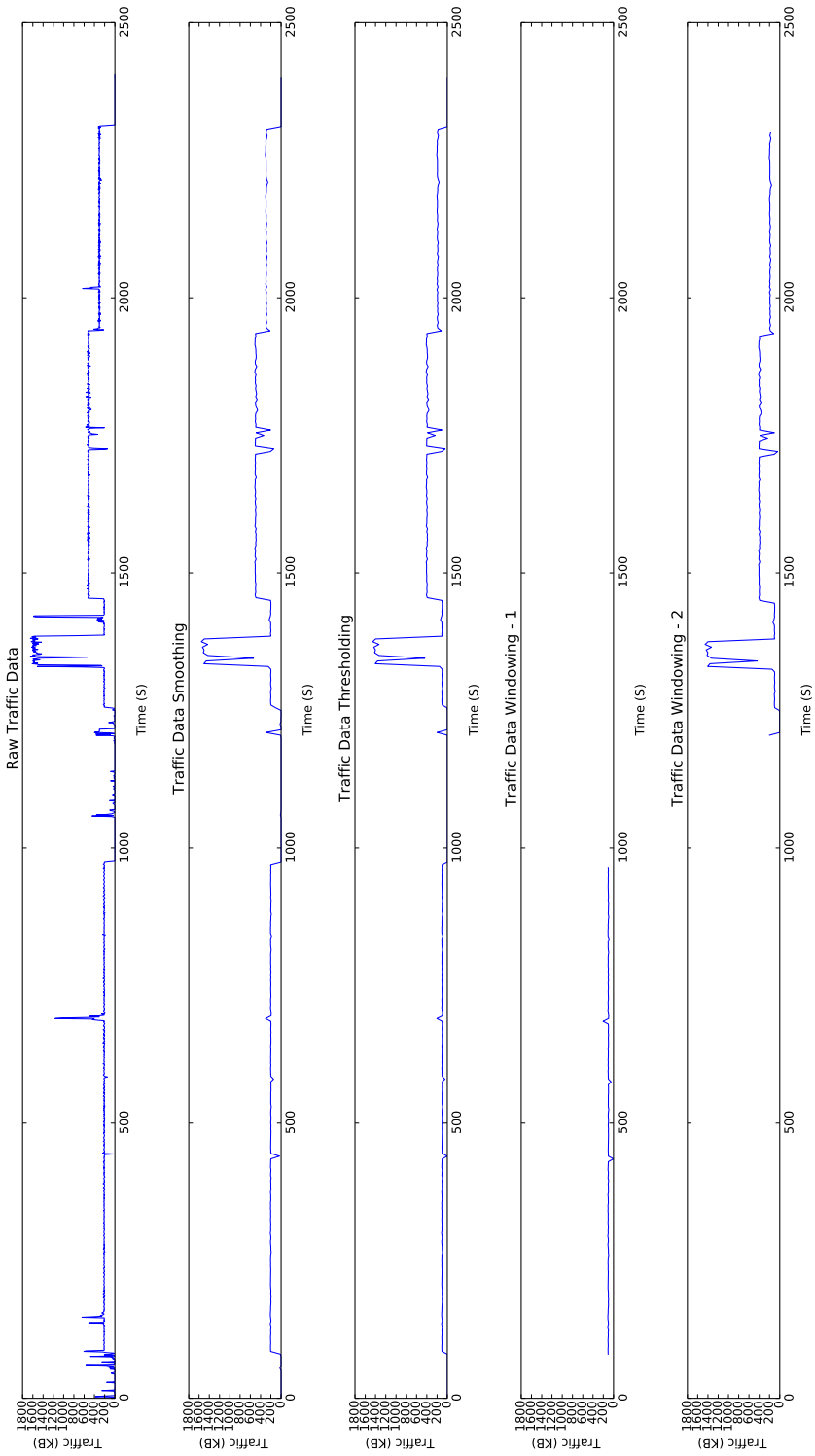


Figure 5.9: Smoothing-Thresholding-Windowing

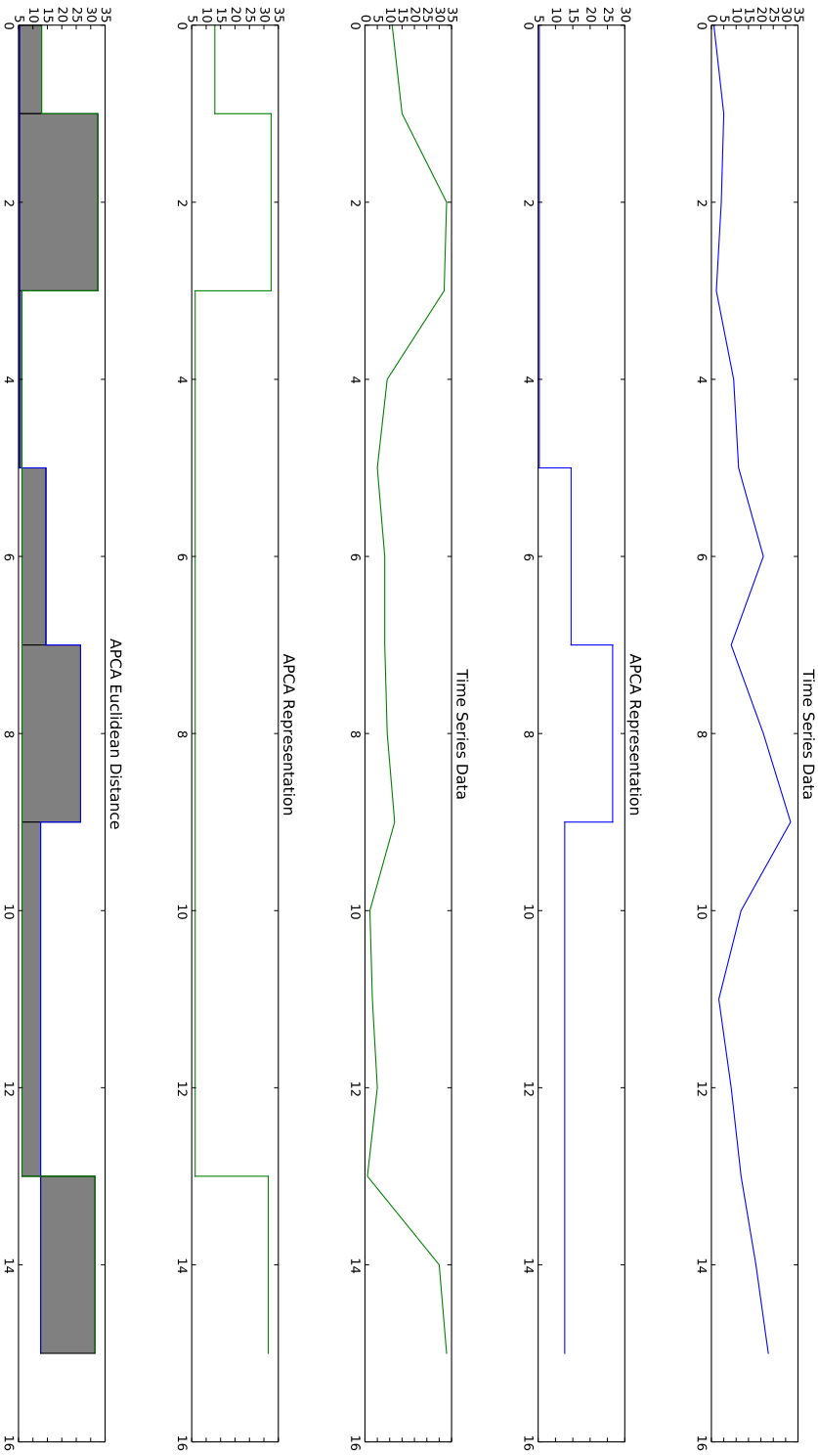


Figure 5.10: APCA Euclidean distance

Chapter 6

Conclusions

Software Defined Networking (SDN) enables a network to be intelligently and centrally controlled via forwarding table entries. The structure and functionality of FTE are more complicated than the ACL and IP routing entry used in traditional networking. Thus, it is vital to have a representation to facilitate the investigation and use of FTE in SDN.

After evaluating all the existing mathematical/logical representations, Boolean algebra is chosen and extended to explore SDN applications. More specifically, the applications such as equivalent forwarding set evaluation, traffic estimation and heavy-traffic flow detection are implemented and verified.

6.1 Review

The contents of this thesis are summarised as below:

- In chapter 3, the process of evaluating equivalence between two forwarding sets in terms of networking functionality is formalised by converting multiple tables and single table into a uniform representation called equivalent forwarding sets. This approach facilitates the flow table management in the controller where a single large

table is usually maintained as well as the flow table deployment in switches where multiple flow tables are more often chosen to adapt to switches' various forwarding pipeline designs.

- In chapter 4, a novel traffic estimation approach based on the existing FTEs' statistics is proposed and evaluated. With the help of Boolean algebra, the deterministic confidence interval of any given flowset can be estimated. It is a non-invasive approach because this process does not require installing any new FTE, which makes it more practicable than the current SDN traffic monitoring solutions.

- In chapter 5, a heavy hitter detection and identification in SDN is implemented. This approach relies on mining traffic statistics and FTE to detect and identify heavy hitters. The experiments on a test bed show that this approach simultaneously achieves considerable accuracy and good scalability. This non-invasive anomaly detection approach enables an SDN controller to identify heavy traffic flows only based on readily available information, which makes this solution easy to deploy on a production network.

6.2 Implications

The study of forwarding table entries mainly focuses on the applications of network engineering. However, it still shows some theoretical implications.

The previous research of SDN representation showed that it was possible to abstract the networking attributes with some adaptation and extension of existing logical and mathematical approaches. However, they rarely delve into the internal structure of forwarding table entries. This thesis examines the individual attributes of forwarding table entries as well as the existing approaches to represent them. More importantly, the pros and cons of these approaches are investigated and the rationale be-

hind them are also explained. Besides this, Boolean algebra has been modified to support wildcard, priority and multiple tables. All these changes make Boolean algebra more applicable to attack the realistic network engineering problems and serve more SDN applications.

The practical implications of the study can be seen from the following three perspectives. First, the equivalent forwarding set evaluation is an effective tool for a network administrator to verify and optimise FTE placement. The truth-table based ACA approach is easily programmed and implemented in real world to check against the various types of forwarding tables. Moreover, it can also be used to detect the routing problems such as loopback and redundancies.

Second, the proposed traffic estimation approach provides an alternative method to monitor a network's running status. This approach is very practical due to its non-invasiveness. Since this approach avoids the additional FTE installation, it will not impact the existing packet forwarding functionality and performance. However, it also has a obvious limitation: the estimation accuracy will be deteriorated with decreasing number of FTEs. Thus, this proposed solution is very suitable for a rough estimation of a network's overall statistics rather than an accurate measurement of a specific flow.

Third, the heavy hitter detection offers an novel way to do traffic analysis which cannot be achieved in traditional networking. SDN has the ability to collect and analyse the real time traffic from a global perspective. It inspires the idea to design a heavy hitter detection based on data mining and FTE correlation. This solution is easily applied in large-scale data-centre since all computation occurs on the SDN controller. The performance will be improved with the scale growth of a network. With the involvement of more switches and FTEs, the characteristics of a suspected heavy flow can be further narrowed down because they rely on the AND ($\wedge$) operations upon the match fields of all correlated FTEs.

In a word, all of these aforementioned practical implications demon-

strate the possibility of developing novel solutions to traditional problems due to SDN's logically centralised architecture.

6.3 Future Work

- **Formal representation of network-wide FTE:** The current FTE representations still focus on the forwarding tables of individual switches. These representations are unlikely to be applicable to a network directly. This shortcoming could be mitigated by introducing formal logics and semantics to represent the connections between switches. Ideally an appropriate FTE representation is capable of covering FTE's three aspects: i) attributes inside a single forwarding table, for example, wildcards, priority; ii) relations among multiple tables, such as the "goto table" attributes; iii) network-wide topology information. Only the representation covering all these afore-mentioned aspects is able to reflect the characteristics of a network and further verify the correctness of FTE deployment or even predict the packet forwarding behaviour. These first two aspects have been well developed in this thesis but it would be better if the "group-table" attribute could be covered due to its functionality in multicast and load balancing. The topology in the third aspect has been integrated into the application of heavy hitter detection but the representation of topology together with other attributes have not been fully exploited.

- **Applications of network-wide FTE representation:** Even though it has been emphasised that a successful FTE representation should cover all the essential attributes, the criteria to evaluate an FTE representation are not only determined by the number of attributes. Their feasibility and usability should be validated in practice, specifically, more work can be done in these two areas: i) verification and optimization of network-wide FTE deployment; ii) traffic mon-

itoring and analysis.

The detection such as the redundancy and routing loop on a single table or individual switch has been largely exploited, however, rare works have been done on the network-wide FTEs. The future works in this area include the verification of FTE deployments such as conversion from a single large table on controller to multiple tables on multiple switches. More importantly, it should be able to direct a networking administrator to optimise FTE deployment to reduce the FTE storage space and even speed up the packet forwarding.

Another potential research point of network-wide FTE representation is traffic monitoring and analysis . A benefit of SDN's centralised architecture is the ability to coordinate all the switches to deliver the packets more economically and efficiently. A good FTE representation should be able to integrate with other non-intrinsic FTE attributes such as topology and statistics. Thus, the applications based on this FTE representation along with the topology and statistic information can assist a network administrator to better understand the real-time status of a network. Moreover, these application should also help an administrator to dynamically analyse, predict and regulate the packet forwarding behaviour inside a software defined network.

Bibliography

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

[2] ONF, "Specification, OpenFlow Switch, Version 1.5.0." `https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf`, 2014.

[3] R. Bifulco and F. Schneider, "OpenFlow rules interactions: definition and detection," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–6, IEEE, 2013.

[4] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cacheflow in software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 175–180, ACM, 2014.

[5] N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson, "A coalgebraic decision procedure for NetKAT," 2014.

[6] M.-K. Shin, M. Kang, J.-Y. Choi, and K.-H. Nam, "Formal Specification for Software-Defined Networks (SDN)." `https://tools.ietf.org/html/draft-shin-sdn-formal-specification-01`, 2013.

[7] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ACM SIGPLAN Notices*, vol. 46, pp. 279–291, ACM, 2011.

[8] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 217–230, 2012.

[9] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model checking invariant security properties in OpenFlow," in *Communications (ICC), 2013 IEEE International Conference on*, pp. 1974–1979, IEEE, 2013.

[10] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, pp. 37–44, ACM, 2010.

[11] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 79–84, ACM, 2012.

[12] A. Wang, S. Moarref, B. T. Loo, U. Topcu, and A. Scedrov, "Automated synthesis of reactive controllers for software-defined networks," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pp. 1–6, IEEE, 2013.

[13] A. Guha, M. Reitblatt, and N. Foster, "Formal foundations for software defined networks," *Open Net Summit*, 2013.

[14] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, and Others, "A NICE Way to Test OpenFlow Applications.," in *NSDI*, vol. 12, pp. 127–140, 2012.

[15] H. Pan, H. Guan, J. Liu, W. Ding, C. Lin, and G. Xie, "The FlowAdapter: Enable flexible multi-table processing on legacy hardware," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 85–90, ACM, 2013.

[16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 351–362, 2011.

[17] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM, 2013 Proceedings IEEE*, pp. 545–549, IEEE, 2013.

[18] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," 2014.

[19] B. Yan, Y. Xu, H. Xing, K. Xi, and H. J. Chao, "CAB: a reactive wildcard rule caching system for software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 163–168, ACM, 2014.

[20] M. Moshref, M. Yu, A. B. Sharma, and R. Govindan, "Scalable Rule Management for Data Centers.," in *NSDI*, pp. 157–170, 2013.

[21] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, "Maple: Simplifying SDN programming using algorithmic policies," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 87–98, ACM, 2013.

[22] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 13–24, ACM, 2013.

[23] D. Williams and H. Jamjoom, "Cementing high availability in OpenFlow with RuleBricks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 139–144, ACM, 2013.

[24] S. Narayana, M. Tahmasbi, J. Rexford, and D. Walker, "Compiling path queries," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pp. 207–222, 2016.

[25] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Practical Aspects of Declarative Languages*, pp. 235–249, Springer, 2011.

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

[27] A. Guha, M. Reitblatt, and N. Foster, "Machine-verified network controllers," in *ACM SIGPLAN Notices*, vol. 48, pp. 483–494, ACM, 2013.

[28] J. Liang, Z. Lin, and Y. Ma, "OF-NEDL: an openflow networking experiment description language based on XML," in *Web Information Systems and Mining*, pp. 686–697, Springer, 2012.

[29] T. De Paula, C. Esteve Rothenberg, M. A. Silva Santos, and L. Bernardes De Paula, "Towards Semantic Network Models via Graph Databases for SDN Applications," in *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, pp. 49–54, IEEE, 2015.

[30] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The design and implementation of open vswitch.," in *NSDI*, vol. 15, pp. 117–130, 2015.

[31] E. Haleplidis, J. H. Salim, S. Denazis, and O. Koufopavlou, "Towards a network abstraction model for sdn," *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 309–327, 2015.

[32] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, "Opflex control protocol," *IETF, Apr*, 2014.

[33] T. Benson, A. Akella, and D. A. Maltz, "Mining policies from enterprise network configuration," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, (New York, NY, USA), pp. 136–142, ACM, 2009.

[34] N.-F. Huang and S.-M. Zhao, "A novel ip-routing lookup scheme and hardware architecture for multigigabit switching routers," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1093–1104, Jun 1999.

[35] B. Pfaff, B. Heller, D. Talayco, D. Erickson, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Pettit, K. Yap, M. Casado, and Others, "OpenFlow Switch Specification," 2009.

[36] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful sdn data planes," *IEEE Communications Surveys Tutorials*, vol. 19, pp. 1701–1725, thirdquarter 2017.

[37] D. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-based access control*. Artech House, 2003.

[38] J. Barkley, "Comparing simple role based access control models and access control lists," in *Proceedings of the second ACM workshop on Role-based access control*, pp. 127–132, ACM, 1997.

[39] B. W. Lampson, "Protection," *SIGOPS Oper. Syst. Rev.*, vol. 8, pp. 18–24, jan 1974.

[40] J. Michaelis and C. Diekmann, "Routing," *Archive of Formal Proofs*, aug 2016.

[41] ONF, "Specification, OpenFlow Switch, Version 1.1.0." `http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf`, 2011.

[42] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, (New York, NY, USA), pp. 99–110, ACM, 2013.

[43] L. Yang, B. Ng, and W. K. G. Seah, "Heavy hitter detection and identification in software defined networking," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–10, Aug 2016.

[44] L. Yang, B. Ng, W. K. G. Seah, and L. Groves, "Equivalent forwarding set evaluation in software defined networking," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 576–579, May 2017.

[45] L. Yang, B. Ng, W. K. Seah, and L. Groves, "Deterministic confidence interval estimation of networking traffic in sdn," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 120–127, IEEE, 2017.

[46] S. Pozo, R. Ceballos, and R. M. Gasca, "Afpl, an abstract language model for firewall acls," in *International Conference on Computational Science and Its Applications*, pp. 468–483, Springer, 2008.

[47] M. K. Wong, Y. V. Gajjar, and R. Kumar, "Efficient acl lookup algorithms," Oct. 5 2010. US Patent 7,808,929.

[48] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, (New York, NY, USA), pp. 13–24, ACM, 2013.

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

[50] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 91–96, ACM, 2013.

[51] C. Marsh, "A Generalized Algorithm for Flow Table Optimization." `http://www.crmarsh.com/static/pdf/Charles_Marsh_SDN.pdf`, 2015.

[52] Y. Bertot, "A short presentation of coq.," in *TPHOLs*, vol. 8, pp. 12–16, Springer, 2008.

[53] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "Netkat: Semantic foundations for networks," in *ACM SIGPLAN Notices*, vol. 49, pp. 113–126, ACM, 2014.

[54] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing rules placement in OpenFlow networks: trading routing for better efficiency," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*, 2014.

[55] J. E. Whitesitt, *Boolean algebra and its applications*. Courier Corporation, 1995.

[56] E. F. Codd, *Relational completeness of data base sublanguages*. IBM Corporation, 1972.

[57] D. Kozen, "Kleene algebra with tests," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 19, no. 3, pp. 427–443, 1997.

[58] E. M. Clarke, "Lecture 1: Propositional Logic." `https://www.cs.cmu.edu/~emc/15414-f12/lecture/propositional_logic.pdf`, 2012.

[59] J. Ullman, "Chapter 14 Predicate Logic." `http://infolab.stanford.edu/~ullman/focs/ch14.pdf`, 1994.

[60] E. A. Emerson, "Temporal and modal logic.," *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, vol. 995, no. 1072, p. 5, 1990.

[61] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, pp. 254–265, ACM, 2011.

[62] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 7–12, ACM, 2012.

[63] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, p. 3, USENIX Association, 2010.

[64] Y. Afek, A. Bremler-Barr, and L. Schiff, "Ranges and cross-entrance consistency with openflow," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 233–234, ACM, 2014.

[65] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "Software transactional networking: Concurrent and consistent policy composition,"

in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 1–6, ACM, 2013.

[66] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 49–54, ACM, 2013.

[67] P. Perešíni, M. Kuzniar, N. Vasić, M. Canini, and D. Kostiū, "Of. cpp: Consistent packet processing for openflow," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 97–102, ACM, 2013.

[68] Y. Yuan, F. Ivančić, C. Lumezanu, S. Zhang, and A. Gupta, "Generating consistent updates for software-defined network configurations," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 221–222, ACM, 2014.

[69] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules placement problem in openflow networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, pp. 1273–1286, Secondquarter 2016.

[70] H. Zhang, C. Lumezanu, J. Rhee, N. Arora, Q. Xu, and G. Jiang, "Enabling layer 2 pathlet tracing through context encoding in software-defined networking," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN*, vol. 14, pp. 169–174, 2014.

[71] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 25–30, ACM, 2013.

[72] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat, "Libra: Divide and conquer to verify forwarding tables in huge networks," in *Proceedings of NSDI*, vol. 14, pp. 87–99, 2014.

[73] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proc. ACM Symposium on SDN Research (SOSR)*, 2016.

[74] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An Industrial-Scale Software Defined Internet Exchange Point," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, (Santa Clara, CA), pp. 1–14, USENIX Association, mar 2016.

[75] C. Yu, C. Lumezanu, H. V. Madhyastha, and G. Jiang, "Characterizing Rule Compression Mechanisms in Software-Defined Networks," in *International Conference on Passive and Active Network Measurement*, pp. 302–315, Springer, 2016.

[76] Y. Wang, J. Bi, P. Lin, Y. Lin, and K. Zhang, "SDI: a multi-domain SDN mechanism for fine-grained inter-domain routing," *Annals of Telecommunications*, pp. 1–13, 2016.

[77] B. Stephens, A. L. Cox, and S. Rixner, "Scalable Multi-Failure Fast Failover via Forwarding Table Compression," *SOSR. ACM*, 2016.

[78] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for {SDN}," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 99–110, ACM, 2013.

[79] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[80] P. Phaal and M. Lavine, "sFlow Version 5." `http://sflow.org/sflow_version_5.txt`, jul 2004.

[81] Chakchai So-In and C. So-In, "A Survey of Network Traffic Monitoring and Analysis Tools," *Cse 576m computer system analysis project, Washington University in St. Louis*, 2009.

[82] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An FPGA-based network intrusion detection architecture," *IEEE Trans on Information Forensics and Security*, vol. 3, no. 1, pp. 118–132, 2008.

[83] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 29–42, 2013.

[84] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: Towards Programmable Network Measurement," *IEEE/ACM Trans. Netw.*, vol. 19, pp. 115–128, feb 2011.

[85] L. Jose, M. Yu, and J. Rexford, "Online Measurement of Large Traffic Aggregates on Commodity Switches.," in *Hot-ICE*, 2011.

[86] M. Malboubi, L. Wang, C. N. Chuah, and P. Sharma, "Intelligent SDN based traffic (de)Aggregation and Measurement Paradigm (iSTAMP)," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 934–942, apr 2014.

[87] Y. Yu, C. Qian, and X. Li, "Distributed and Collaborative Traffic Monitoring in Software Defined Networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), pp. 85–90, ACM, 2014.

[88] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, (New York, NY, USA), pp. 25–30, ACM, 2013.

[89] T. S. Chen, D. Y. Lee, T. T. Liu, and A. Y. Wu, "Dynamic reconfigurable ternary content addressable memory for openflow-compliant low-power packet processing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, pp. 1661–1672, Oct 2016.

[90] "OpenFlow Switch Specification, Version 1.5.1." `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf`.

[91] ONF, "The Benefits of Multiple Flow Tables and TTPs, Version Number 1.0." `https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/TR_Multiple_Flow_Tables_and_TTPs.pdf`.

[92] F. M. Brown, *Boolean reasoning: the logic of Boolean equations*. Springer Science & Business Media, 2012.

[93] The Linux Foundation, "Open vSwitch, An Open Virtual Switch," 2017.

[94] P. L. Philippe Biondi Guillaume Valadon, "Scapy." `http://www.secdev.org/projects/scapy/`.

[95] A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S. Chen, "Umon: Flexible and fine grained traffic monitoring in open vswitch," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, (New York, NY, USA), pp. 15:1–15:7, ACM, 2015.

[96] M.-H. Wang, S.-Y. Wu, L.-H. Yen, and C.-C. Tseng, "Pathmon: Path-specific traffic monitoring in openflow-enabled networks," in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 775–780, July 2016.

[97] V. J. Easton and J. H. McColl, "Statistics Glossary v1. 1," 1997.

[98] NTT, "Ryu SDN Framework." `https://osrg.github.io/ryu/`.

[99] K.-C. Lan and J. Heidemann, "On the correlation of internet flow characteristics," Tech. Rep. ISI-TR-574, Information Sciences Institute, University of Southern California, jul 2003.

[100] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the Characteristics and Origins of Internet Flow Rates," in *Proc of ACM SIGCOMM*, (Pittsburgh, Pennsylvania, USA), pp. 309–322, 2002.

[101] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 137–148, ACM, 2003.

[102] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 155–166, ACM, 2004.

[103] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications," in *Proc of the 4th ACM SIGCOMM Internet Measurement Conference (IMC)*, (Taormina, Sicily, Italy), pp. 101–114, 2004.

[104] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc of the 9th ACM SIGCOMM Internet Measurement Conference*, (Chicago, IL, USA), 2009.

[105] W. Willinger, "The discovery of self-similar traffic," in *Performance Evaluation: Origins and Directions*, pp. 513–527, Springer, 2000.

[106] A. Vahdat, M. Al-Fares, N. Farrington, R. N. Mysore, G. Porter, and S. Radhakrishnan, "Scale-Out Networking in the Data Center," *IEEE Micro*, vol. 30, no. 4, pp. 29–41, 2010.

[107] S. Muthukrishnan, *Data streams: Algorithms and applications*. Now Publishers Inc., 2005.

[108] R. Agrawal, C. Faloutsos, and A. N. Swami, "Efficient Similarity Search In Sequence Databases," in *Proc of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO)*, (Chicago, Illinois, USA), pp. 69–84, 1993.

[109] E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrotra, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Trans on Database Systems*, vol. 27, pp. 188–228, jun 2002.

[110] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[111] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot, "A pragmatic definition of elephants in internet backbone traffic," in *Proc of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, (Marseille, France), pp. 175–176, 2002.

[112] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc of the USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, (San Jose, CA), 2012.

[113] R. Edwards, Y. Ye, A. Kaul, and J. Yu, "Characterization of {SDN} switch response time in proactive mode," in *Proc of the Open Networking Summit (ONS)*, (Santa Clara, CA, USA), 2014.

[114] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran, "Reformulating the Monitor Placement Problem: Optimal Network-wide Sampling," in *Proc of the 2nd ACM Conference on Future Net-*

working Technologies (CoNEXT), (Lisboa, Portugal), pp. 5:1—-5:12, 2006.

[115] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection," in *Proc of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS)*, (San Jose, California, USA), pp. 93–102, 2006.

[116] A. K. Chandra and G. Markowsky, "On the number of prime implicants," *Discrete Mathematics*, vol. 24, no. 1, pp. 7–11, 1978.

Appendices

Appendix A

Boolean Reasoning

An n-variable function $f : \mathrm{B}^n \to \mathrm{B}$ is called a *Boolean function* if and only if it can be expressed by a Boolean formula.

For any element $b \in \mathrm{B}$, the $constant function$, defined by $f(x_1, x_2, ..., x_n) = b \quad \forall (x_1, x_2, ..., x_n) \in \mathrm{B}^n$,
is an n-variable Boolean function.

Product Term: The product term of a n-variable Boolean function is defined to be the logical **AND** of some or all of these variables.

Minterm (Canonical) Form: A product term in which all variables appear once and only once, i.e., a function with n variables has 2^n possible unique minterms.

Sum of Product (SOP): SoP is a disjunction (OR) of minterms. A SOP of a Boolean function is a sum(OR) of the minterms which represent all the "true" values in this function.

Equivalent($\equiv$): Two formulas will be called $equivalent(\equiv)$ in case they represent the same Boolean function, i.e., in case one can be transformed into the other by the application of the rules of Boolean algebra.

Congruent($\cong$): Two formulas in the form of disjunctive normal form (DNF) is *congruent* in case that one can be transformed into the other using only the commutative rule.

Included($\leq$): Given two Boolean function g and h, if the identify $gh' =$

0 is satisfied, then g is *included* in h, written $g \leq h$.

Implicant / Prime Implicant: An *implicant* of a Boolean function f is a term p such that $p \leq f$. Any term of an DNF formula for f is clearly an implicant of f. An implicant p of f is a *prime implicant* if and only if no other implicant of f contains p, i.e., for any term q, $p \leq q \leq f \rightarrow p = q$.

Absorptive: An DNF formula F will be called *absorptive* in case no term in F is absorbed by any other term in F. The equivalent absorptive formula (ABS) of a non-absorptive F can be obtained by successive deletion of terms absorbed by other terms in F.

Syllogistic: A formula F will be called *syllogistic* in case, for every DNF formula G, $G \leq F \rightarrow G \ll F$.

Blake Canonical Form (BCF): Let F be a syllogistic formula for a Boolean function f. The formula ABS(F), the *Blake canonical form* for f, which is denoted by *BCF(f)*. If a DNF formula F is not syllogistic, it contains terms p and q, having exactly one opposition, such that $c(p, q)$ is not formally included in F. F is *syllogistic* if and only if every prime implicant of f is a term of F.

Appendix B

Boolean Function Operations on Match Fields

Let M_0 and M_1 denote a m-dimension match set $M_0 = \{f_0^0 \wedge f_1^0 \ldots \wedge f_m^0\}$ and n-dimension match set $M_1 = \{f_0^1 \wedge f_1^1 \ldots \wedge f_n^1\}$ respectively. The operations between M_0 and M_1 are defined as follows:

Definition B.0.1 (Exact matching)**.** The physical meaning of two *exactly matching* FTEs is that one's matching flow will also match another one.

$$M_0 = M_1 : \ \forall i \ \exists j \mid [(f_i^0 = f_j^1) \wedge (f_i^1 = f_j^0)],\ f_{i,j}^0 \in M_0 \wedge f_{i,j}^1 \in M_1$$

Definition B.0.2 (Subset (Superset))**.** The physical meaning of $M_0 \subset M_1$ is that M_0's matching flow does match M_1, but not vice versa. M_1 is also called a *superset* of M_0.

$$M_0 \subset M_1 : \ \forall i \ \exists j \mid [(f_i^0 \subset f_j^1) \vee (f_i^0 = f_j^1)],\ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Definition B.0.3 (Disjoint ($\bowtie$))**.** The physical meaning of two *disjoint* FTEs is that one's matching flow will *NOT* match another and their respective

matching packets share no common characteristics.

$$M_0 \overline{\bowtie} M_1 : \ \forall i,j \mid f_i^0 \wedge f_j^1 = \phi,\ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Definition B.0.4 (Joint ($\bowtie$))**.** The physical meaning of two *joint* FTEs is that their matching packets share one or more common characteristics, e.g., in the same IP subnet.

$$M_0 \ \bowtie \ M_1 : \ \exists i,j \mid f_i^0 \wedge f_j^1 \neq \phi,\ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Three logical operations are used on match sets in the HH identification module (see Section 5.3): (i) Negation (not): $\neg$; (ii) Conjunction (and): $\wedge$; and (iii) Disjunction (or): $\vee$.

Negation is used in the set theory context and defined as follows:

$$\begin{aligned} \neg M_0 &= \neg\{f_0^0 \wedge f_1^0 \ldots \wedge f_m^0\} \\ &= \{\neg f_0^0 \vee \neg f_1^0 \ldots \vee \neg f_m^0\}. \end{aligned}$$

The conjunction operator on match sets M_0 and M_1 implies that packets of $M_0 \wedge M_1$ can match both sets simultaneously (as expressed in (B.1)).

$$M_0 \wedge M_1 = \begin{cases} M_0 \ or \ M_1 & \text{if } M_0 = M_1 \\ M_1 & \text{if } M_0 \subset M_1 \\ M_c\big(M_0^{'} \wedge M_1^{'}\big) & \text{if } M_0 \bowtie M_1 \\ M_0 \wedge M_1 & \text{if } M_0 \ \overline{\bowtie} \ M_1 \end{cases} \tag{B.1}$$

where M_c is the greatest common subset of M_0 and M_1, $M_0^{'}$ and $M_1^{'}$ are the subsets of M_0 and M_1, respectively, from which M_c has been removed.

The disjunction operator on match sets asserts that the matching packets

of $M_0 \vee M_1$ match either M_0, M_1, or both as expressed in (B.2).

$$M_0 \vee M_1 = \begin{cases} M_0 \ or \ M_1 & \text{if } M_0 = M_1 \\ M_0 & \text{if } M_0 \subset M_1 \\ M_c\big(M_0^{'} \vee M_1^{'}\big) & \text{if } M_0 \bowtie M_1 \\ M_0 \vee M_1 & \text{if } M_0 \overline{\bowtie} M_1 \end{cases} \tag{B.2}$$

Appendix C

Formula vs Truth-table Comparison

There are two approaches to evaluate the equivalence of two Boolean functions: formula based and truth table based comparison. The former method converts a Boolean function formula into a normal form while the latter transforms all the "truth" value in the Boolean function into a table in which each row represents exact one value. In both methods, the complexity of comparison depends on two factors: width and length. The width indicates the number of literals in a formula or the bits of a value in a truth table and the length represents the number of products in a disjunctive normal form (DNF) or values in a truth table.

An *n-variable Boolean system* on a Boolean algebra $\mathbb{B}$ is a collection of simultaneously-asserted equations, as follows:

$$\begin{aligned} p_1(X) &= q_1(X) \\ &\vdots \\ p_k(X) &= q_k(X) \end{aligned}$$

The p_i and q_i are n-variable Boolean functions on $\mathbb{B}$; X denotes the vector $(x_1, x_2, ..., x_n)$. The Boolean system is equivalent to the single equation

$$f(X) = 0,$$

where f is defined by

$$f = \sum_{i=1}^{k} (p_i \oplus q_i)$$

Thus, for any two forwarding sets F_g and F_h, their equivalence is determined by the equivalence of every Boolean function g_i and h_i, where g_i and h_i represent the Boolean function for the same action set a_i in F_g and F_h, respectively.

It is obvious that any two Boolean functions are equivalent if they comprise the same DNF expression. The method based on the comparison of formula is also called *formula based equivalence*. One major challenge of this method is that how to transform a Boolean function into a uniform formula. The formula must have one property: it has a consistent form for any two equivalent Boolean functions. One solution is sum of products (SOP) form in which all "truth" values of a Boolean function represent by a minterm product. However, the size of SOP is the largest among all disjunctive form. The *Blake canonical form* (see Appendix A) of a Boolean function f (denoted by $BCF(f)$), is minimal within the class of syllogistic formulas while also comprises the disjunctions of all of the prime implicants of f. Hence, the BCF of any two equivalent Boolean functions f and g must be identical or congruent, i.e., $BCF(f) = BCF(g)$ or $BCF(f) \cong BCF(g)$.

Another way to express a Boolean function is using a truth table in which each row represents a minterm canonical form (*truth-table based equivalence*). If a Boolean function f has k elements, then the maximum number of rows in its truth table is 2^k. The equivalence of two Boolean functions based on the truth-table comparison is a straightforward but time-consuming process, which is an exclusive OR bitwise operation of all entries in these two tables.

The width of a Boolean function for a forwarding set is determined by

the total number of bits in all match fields. In a formula-based expression, each literal represents a single bit, thus the width also determines the number of literals of all products in a Boolean formula. The number can be further reduced with the introduction of "do not care" bits. If the "do not care" bit exists in the same position for all match fields, it can be removed by the "absorptive" operation. However, the truth-table representation does not benefit from this. It must be expressed using the same width as the original Boolean function. This is because a literal in a formula represents both the value of a bit but also the position of this bit. Take the function in Eq.(C.1) as an example,

$$f = a^{'}b^{'}cde^{'} + a^{'}bcde^{'} + ab^{'}cde^{'} + abcde, \tag{C.1}$$

if the literal c in (C.1) is set to "do not care", then its formula based expression simplifies to:

$$f = a^{'}b^{'}de^{'} + a^{'}bde^{'} + ab^{'}de^{'} + abde,$$

in which the literal c is omitted because all the value in its specified position is "do not care". However, in the truth table representation Eq.(C.2), the third bit which is represented by the literal c cannot be removed from the truth table, otherwise, the position of c cannot be inferred.

$$f = \begin{bmatrix} 0 & 0 & * & 1 & 0 \\ 0 & 1 & * & 1 & 0 \\ 1 & 0 & * & 1 & 0 \\ 1 & 1 & * & 1 & 1 \end{bmatrix}. \tag{C.2}$$

The minimum length of a Boolean function is equal to the least number of products of its equivalent absorptive formula (ABS). However, the ABS is not identical for any two given equivalent Boolean function thus it cannot be used to evaluate the equivalence of any two Boolean functions.

Another form, the BCF formula, can be used to compare two functions. Thus the number of products in the BCF formula of a Boolean function is more valued than the minimum ABS during the evaluation of equivalence.

The $BCF(f)$ is the disjunction of all of the prime implicants of f. It has been shown in [116] that the prime implicants are related to the number of conjunctions and variables

$$\max\left(3^{\lfloor k/3 \rfloor}, 3^n/n\right) \leq N_{pi} \leq \min\left(2^k, 3^n/\sqrt{n}\right) \tag{C.3}$$

Thus any complexity reduction on $BCF(f)$ lies upon the decreasing the conjunctions or variables in f.

Appendix D

Test Environment Setup

D.1 Overview

In this thesis, all of the three contributions involve functional tests and performance evaluation. The heavy hitter detection was the first project and conducted on a small-scale test bed which was built on top of physical switches. The rest two contributions were conducted on an Open vSwitch for efficiency and convenience.

The overall architecture of each test environment has already been described in their respective chapters, this appendix illustrates more details on the configurations, commands and source code explanation.

D.2 Open vSwitch on VirtualBox

As depicted in Fig. D.1, an Oracle VirtualBox was installed on a Ubuntu Server. On this VirtualBox, three virtual machines are created. They are names as "VM1", "VM2" and "VM3" respectively. The VirtualBox host, which is the Ubuntu server, also bears the role of OpenFlow controller. All experiments in this thesis choose Ryu as the SDN controller.

The host machine was assigned a public IP address and there was a

domain name (https://sdn.lianghub.com) to associate with it to make it accessible outside of the lab. There virtual Ethernet interfaces were created on the host to connect with the three VMs to make sure they can be accessed by host via SSH.

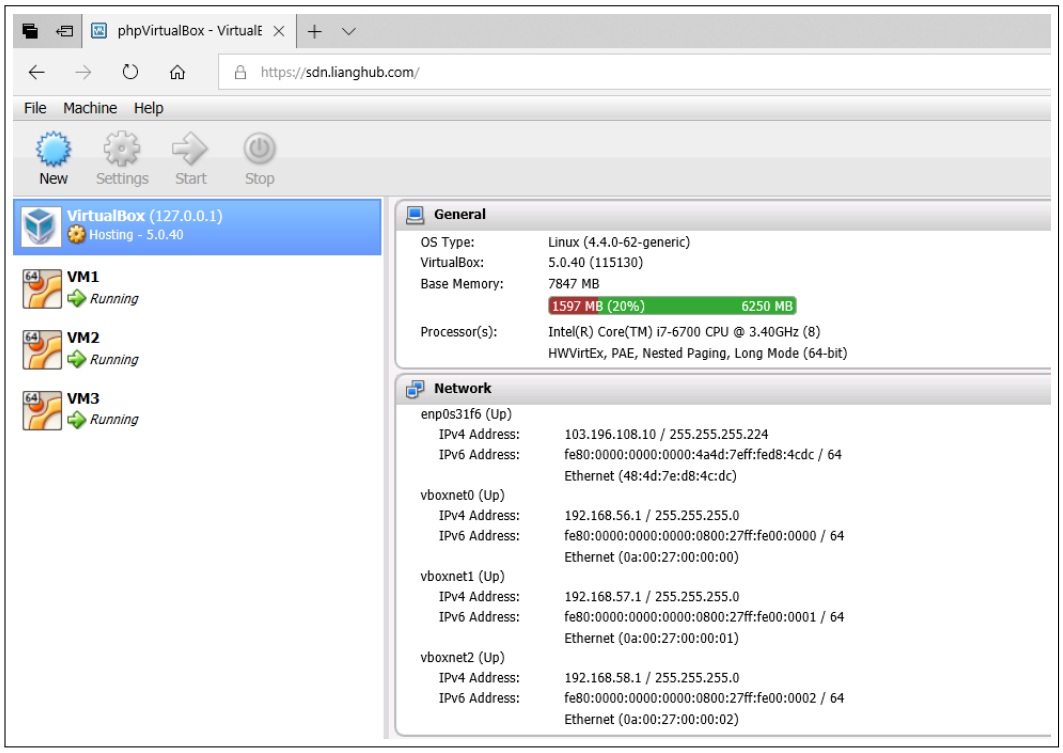


Figure D.1: Snapshot of Open vSwitch on VirtualBox

The Open vSwitch was only installed on VM2 and three additional virtual network adapters was created to connect with the other two VMs as well as the host server. Figure D.2 depicts the details of virtual network adapters' configurations. During the experiment, this VM played the role of OpenFlow switch to receive the commands from the OpenFlow controller (the host server) to install FTEs and report traffic statistics.

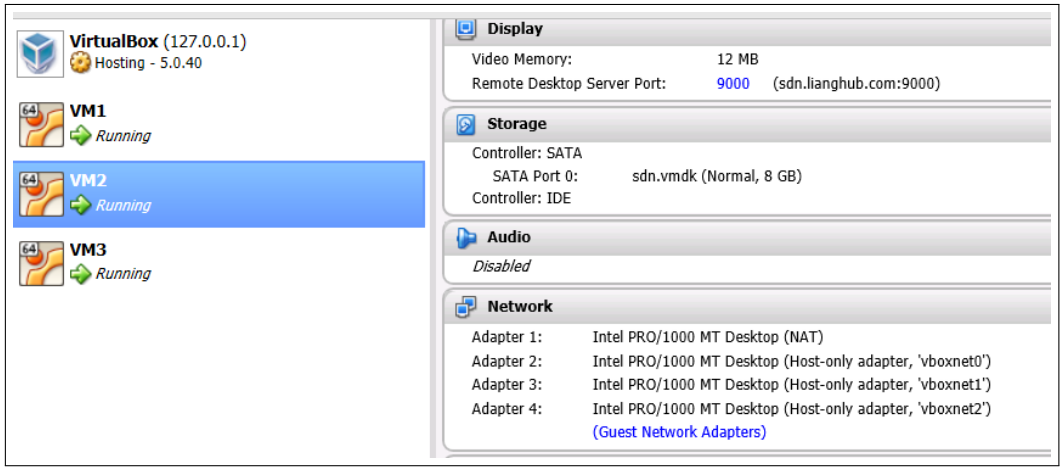


Figure D.2: Configuration for individual VM (VM2)

D.3 OpenFlow Physical Switch Test Bed

In the experiment of heavy hitter detection, a small-scale test bed with physical switch (Pica8, https://www.pica8.com/) was used. To increase the number of end hosts (the machine to send/receive packets to/from switches), virtual machines were built on some machines (physical servers in Fig. D.3). The overall topology of the test bed is illustrated in Fig. D.3 in which data plane and control plane are completely separated. The OpenFlow controller, as well as other switches, were put inside a private network which can be still accessed from WAN via a Network Address Translation (NAT) router.

In this test bed, all the switches came from Pica8. Edgecore Networks switches (https://www.edge-core.com/) were also tested but finally they were not used in the experiment. A major reason was that Edgecore adopted Open Network Linux operating system which offers better flexibility but involves more configuration effort.

Table D.1 listed the most frequently used Pica8 OpenFlow commands in the experiment. In this table, "br0" stands for OpenFlow bridge 0 which defines the OpenFlow port range.

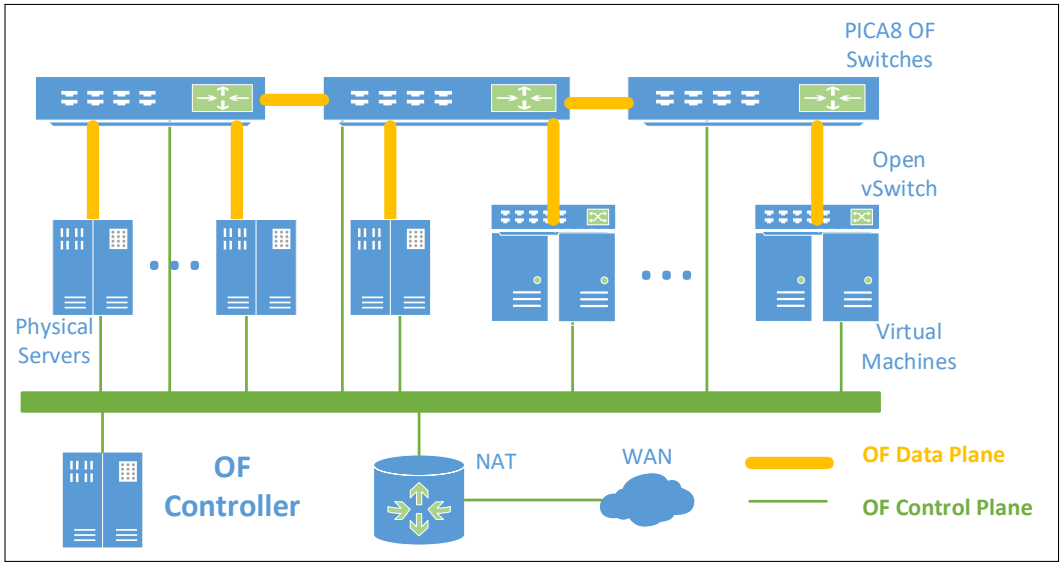


Figure D.3: Topology of OpenFlow physical switch testbed

Table D.1: Most frequently used Pica8 OpenFlow command

Function	Command
Create a bridge	ovs-vsctl add-br br0 – set bridge br0 datapath_type=pica8
Add port to a bridge	ovs-vsctl add-port br0 ge-1/1/1 – set interface ge-1/1/1 type=pica8
Remove port from a bridge	ovs-vsctl del-port br0 ge-1/1/1
Set controller for a bridge	ovs-vsctl set-controller br0 tcp:192.168.138.115:6633

D.4 Scripts and Source Code

During the experiments of equivalent forwarding set evaluation and traffic estimation, a large amount of OpenFlow table entries were created. The following code snippet illustrates how to generate this by a bash script.

```
#!/bin/bash
hexchars="0123456789ABCDEF"

echo $end
for (( c=1; c<=400000; c++ ))
do
end="aa:bb:cc"$( for i in {1..6} ; do echo -n ${hexchars:$(( $RANDOM % 16 )):1}
    ; done | sed -e 's/\(..\)/:\1/g' ) command=$(printf "curl -X POST -d
```

```
    '{\"dpid\": 8796751660512,\"cookie\": 1,\"cookie_mask\": 1,\"table_id\":
    0,\"idle_timeout\": 0,\"hard_timeout\": 0,\"priority\": 22222,\"flags\":
    1,\"match\":{\"dl_dst\": \"%s\"},\"actions\":[{\"type\":\"OUTPUT\",\"port\":
    2}]}' http://103.196.108.10:8080/stats/flowentry/add" "$end")
echo $command >> 400000flow
done
```

In the evaluation of the most effective networking traffic representation (Chapter 5), the algorithm DFT, DWT, PLAN and APCA were implemented by MATLAB. For the heavy hitter detection, the traffic collection and pattern recognition were implemented by Python. All the source code have been uploaded to Github (https://github.com/csliangy/hhdetection).

Appendix E

Acronyms and Abbreviations

ABS ..... Absorptive Formula

ACA ..... Action Oriented Conversion Approach

ACL ..... Access Control List

AMF ..... Actual Match-field

APCA ..... Adaptive Piecewise Constant Approximation

AS^3 ..... Simple-Subset-Superset on AMF

ASIC ..... Applicationspecific Integrated Circuit

BCF ..... Blake Canonical Form

CNF ..... Conjunctive Normal Form

DCI ..... Deterministic Confidence Interval

DECISION ... DEterministic Confidence Interval eStimatION

DFT ..... Discrete Fourier Transform

DHHDS ..... Distance-based HH Detection System

DIP ..... Destination IP

D-ITG ..... Distributed Internet Traffic Generator

DMAC ..... Destination MAC

DNF ..... Disjunctive Normal Form

DWT ..... Discrete Wavelet Transform

EFS ..... Equivalent Forwarding Set

FIB ..... Forwarding Information Base

FTE ..... Forwarding Table Entry

FP ..... False Positive

FPGA ..... Field-programmable Gate Array

HA ..... High Availability

HH ..... Heavy Hitter

HHH ..... Hierarchical Heavy Hitter

IP ..... Internet Protocol

LAN ..... Local Area Network

LODCI ..... Lower Bound of Optimal DCI

LSQ ..... Least-square Method

MAC ..... Medium Access Control

MAF ..... Match-action Form

MFA ..... Match-fields Oriented Conversion Approach

MFT ..... Multiple Flow Table

MHH ..... Multi-dimensional Heavy Hitter

MPLS ..... Multiprotocol Label Switching

ODCI ..... Optimal Deterministic Confidence Interval

OMF ..... Original Match-field

ONF ..... Open Networking Foundation

OS^3 ..... Simple-Subset-Superset on OMF

OTN ..... One-stage to N-stage Flow Tables

PLA ..... Piecewise Linear Approximation

PPV ..... Positive Predictive Value

ProgME ..... Programmable Network MEasurement

RBAC ..... Role-based Access Control

RMT ..... Reconfigurable Match Table

S^3 ..... Simple-Subset-Superset

SCF ..... Specified Common Field

SDN ..... Software Defined Networking

SMAC ..... Source MAC

SOP ..... Sum of Product

ST ..... Similarity Threshold

TM-FTE ..... Traffic Measurement FTE

TM-Table ..... Traffic Measurement Table

TP ..... True Positive

UODCI ..... Upper Bound of Optimal DCI

VLAN ..... Virtual LAN

VM ..... Virtual Machine

WAN ..... Wide Area Network