ISSN 2616-7433 Vol. 2, Issue 11: 138-147, DOI: 10.25236/FSST.2020.021120

Review on the 10-Year Development of Software Defined Networks

Yiming Tang[1], Qiurong Chen[2], Jiayi Xu[3], Yixiao Chen[4]

1 Faculty of Engineering, the University of Sydney, Sydney, Australia
ytan3018@uni.sydney.edu.au
2 Shijiazhaung No.24 High School, Shijiazhuang, China
3 Faculty of Computer Science and Technology, Nanjing Tech University, Nanjing, China
4 Shandong Experimental High School, Jinan, China

ABSTRACT. *Software defined network (SDN) has existed and been under developing for more than a decade. It presents a centralised method to manage networks rather than the distributed traditional way. SDN has become a hot topic and along with such popularity, it has received both constructive advice and pragmatic criticism. This paper evaluates the development of SDN from its birth to recent critiques via reviewing important published papers in the history of SDN, including the invention of SDN, how SDN has been implemented and deployed widely, direction toward which OpenFlow may evolve and the reality of SDN against its hype. Rather than much of detailed elaboration, this paper mainly focuses on the impact on SDN brought by each paper's invention/conclusion.*

KEYWORDS: *SDN, Openflow, B4, Traffic engineering, P4*

1. Introduction

SDN is a new type of network architecture. “Software Defined” means network management in SDN relies on a centralized controller, which is implemented in software, rather than sophisticated routers. SDN separates the control plane and data plane, so as to realize the programmable control of the hardware through the software platform in the centralized controller and fulfill flexible demands in dynamic network. However, a conventionally network management relies on the network equipment, which integrates an operating system and dedicated hardware. SDN is a centralized control network, while traditional networks are distributed network.

1.1 Separation of Control Plane and Data Plane

The responsibility of SDN-based switches, which is in data plane, are forwarding packages based on flow table entries from control plane. Control plane is implement by software, which determines the routing of packets, and send routing result to switches in form of flow table.

1.2 Logically Centralized Control

Controller rather the distributed routers. So that, if the state of network changes, there is no need to configure the devices one by one, and only the controller needs to be configured.

1.3 General Software and Programmable Hardware

OpenFlow has become the mainstream method for network programming using a central controller. The OpenFlow central controller that masters the entire network topology can designate all network switches with OpenFlow Agent.

2. OPENFLOW: THE START OF EVERYTHING

OpenFlow is a protocol to implement SDN. It offers a way for researchers to run experiments and innovate in the traditional network. It is easy to be accepted by the researchers and network vendors since it can not only offer scientists a way to run experiments on heterogeneous switches at line-rate with high port-density, but also do not need the vendors to expose their internal working[1].

2.1 Limits with Experiments

Some network environment limits, like most of the installed base of switches or protocols, create barriers to do researches on them, and as a result, the path to make innovation become more and more difficult. In order to undergo these challenges, the most important thing is how to run experiments without disrupting other users who are dependent on the network. Actually, there are some existing platforms in consideration, but the bad performance, high cost or insufficient network interfaces make these platforms unusable. To solve these problems, OpenFlow should be designed to support various types of researches and experiments, have high performance but cost less, and it needs to isolate experimental traffic from production traffic.

2.2 Openflow Switch

An OpenFlow switch contains a Flow Table that shows the switch how to process, a Secure Channel that allows controllers to send commands and packets to the switch, and the OpenFlow Protocol that allows researchers communicate with the switch. Fig. 1 shows the structure of a basic OpenFlow Switch. The dedicated OpenFlow switch is a dumb data path element which forwards packets between ports. The dedicated OpenFlow must support three actions: (1) forwarding packets to a given port; (2) encapsulating and forwarding the flow’s packets to the controller; (3) dropping packets according to security settings. OpenFlow switches contain the flow-tables that run at line-rate to implement firewalls and collect data. The flow-table is the basis of data forwarding. It holds the network information of each layer in the network, so it can execute more abundant forwarding rules. When the switch receives packets from the host, it will query for corresponding action and the output port on itself. The OpenFlow is the protocol to program the flow-table in different switches and routers. The flow-table consists of three fields: (1) a packet header to define the flow; (2) the action to define how to process; (3) the statistics to keep track of the number of packets and bytes for each flow. The first field usually includes source MAC address, destination MAC address, Ethernet type, VLAN ID, source IP address, destination IP address, IP port, source TCP port, and destination TCP port.

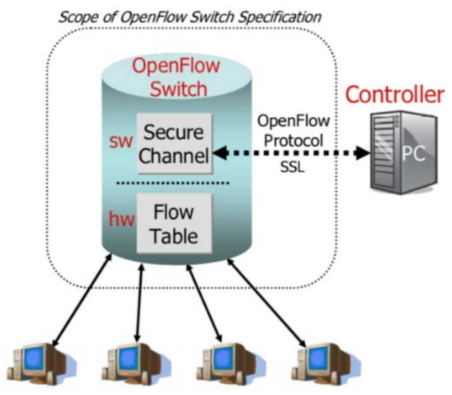

Fig.1 Ideal Openflow Swtich[1]

2.3 Openflow-Enalbed Commercial Switch

Fig. 2 shows a network of OpenFlow enabled commercial switches, all the flow tables are managed by the same controller and the OpenFlow protocol allows several controllers to control a switch together to increase the performance. The OpenFlow enabled switch must isolate experimental traffic from production traffic, so there are some ways to achieve this goal: (1) forward this flow’s packets through the switch’s normal processing pipeline; (2) define separate sets of VLANs for experimental traffic and production traffic.

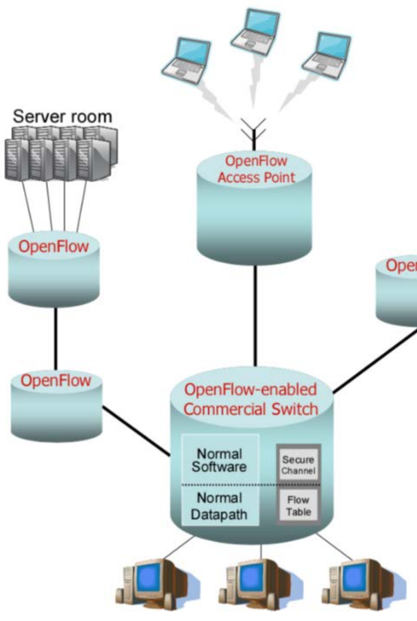

Fig.2 Openflow Enabled Commercial Switches and Routers[1]

2.4 Using Openflow

When a researcher wants to run an experiment in a production network which is used by many other people at the same time, researches want two additional properties for the network: (1) other users except the experiment designer should forward their packets through a standard and tested routing protocol that runs in the switches or routers; (2) the researcher only can add flow entries for the traffic that administrators of the network allow. The first property can be achieved by the OpenFlow enabled switches and the second one depends on the controllers of the network.

3. B4: LESSONS LEARNT FROM THE DEPLOYMENT OF SDN

The previous section talks about the beginning of SDN, a solution for testing new ideas in computer network. However, SDN is not only beneficial for researchers but also for companies. Google's experience with B4, a Globally-deployed Software Defined WAN, strongly proves SDN's value economically. Besides, the building of B4 also provides experience on how to incrementally deploy SDN while traditional network is still a popular trend[2].

3.1 Design Principles of B4

Two principles are followed when Google was designing B4. Firstly, Google accepts failures, which are inevitable, as norm. This is different from traditional designs, which tries to avoid failure, even at a high price. This principle is also used in Google File System. Since the society has entered the era of big data, it is harder and harder to avoid failure in such large and sophisticated system. Therefore, Google concentrates on fixing the mistakes immediately and reducing the side effect of failure. Secondly, Google built their own switch hardware that removes complicated control functionality and exports a simple interface to program forwarding table entries. This is consistent with the separation of control plane and data plane in SDN.

3.2 Design of B4 Architecture and Integrated Routing

(1)Overview of B4 architecture: B4 has sites all over the world and each site has many switches. Therefore, B4 has three layers (shown in Fig. 3): global layer, site controller layer and switch hardware layer. According to the separation of control plane and data plane in SDN, switch hardware layer merely forwards traffic and site controller layer hosts complicated control softwares. One site has different Network Control Servers(NCS) and

each server executes many Network Control Applications(NCA), like OpenFlow Controller, Paxos and Quagga. Hence, each site has multiple available software replicas running on different NCSs. While OpenFlow Agents(OFA) maintain active connections to multiple OpenFlow Controllers(OFC), communication is active to only one OFC at a time and only a single OFC maintains state for a given set of switches. To manage these replicas, one site will elect one of them as the primary instance, using Paxos to set up a leader election. Global layer contains some logically centralized applications, which replicates across multiple WAN sites with separate leader elections. OFA in switches and OFC in NCS are responsible for communication between switch layer and site controller layer.

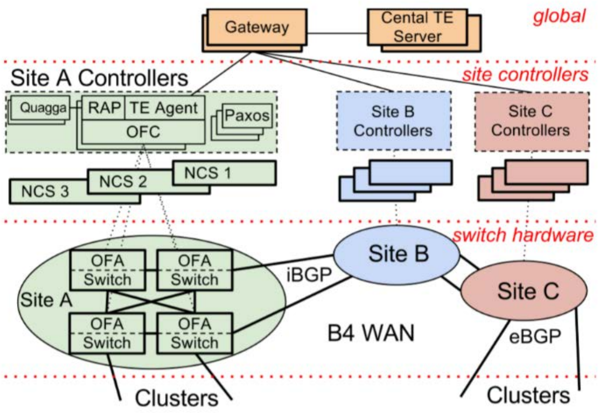

Fig.3 B4 Architecture Overview[2]

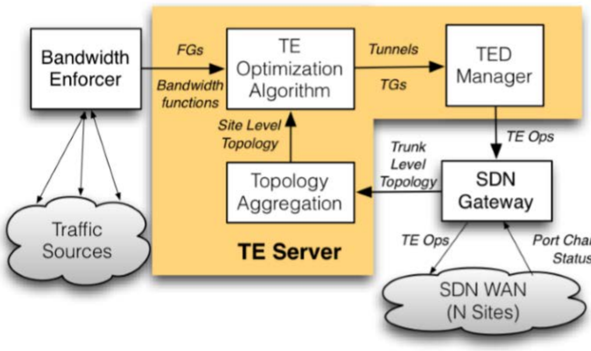

Fig.4 Traffic Engineering Overview[2]

Google chose to deploy routing and traffic engineering as independent services, with the standard routing service deployed initially and central traffic engineering deployed subsequently as an overlay. This is conducive to focus on initial work on SDN infrastructure and debug before implementing Traffic Engineering (TE). Besides, deploying TE as an overlay enables an efficient fault recovery mechanism.

(2)Integrated routing: Google chose the open source Quagga stack for BGP/ISIS on NCS. As shown in Fig. 4, Routing Application Proxy(RAP), an SDN application, provides connectivity between Quagga and OF switches, ensuring that network state in Quagga is up-to-date. After result of Quagga, Routing Information Base(RIB), sends to OFC, Onix converts RIB into Flow table and ECMP group table, which is usable in switches.

3.3 Traffic Engineering on Sdn

(1)Packet routing: The goal of TE is to use max-min fair solution for traffic allocation. TE optimization algorithm (shown in Fig. 5) needs information about both flow (flow table and bandwidth function) and sites (site topology) and caclculates the state for forwarding packets along multiple paths, which is stored in Traffic

Engineering Database (TED manager). The Topology Aggregation abstracts the topology from SDN Gateway and significantly reduces the size of the graph. The Bandwidth Enforcer is responsible for extracting flow table and bandwidth function from grouped applications of a site.

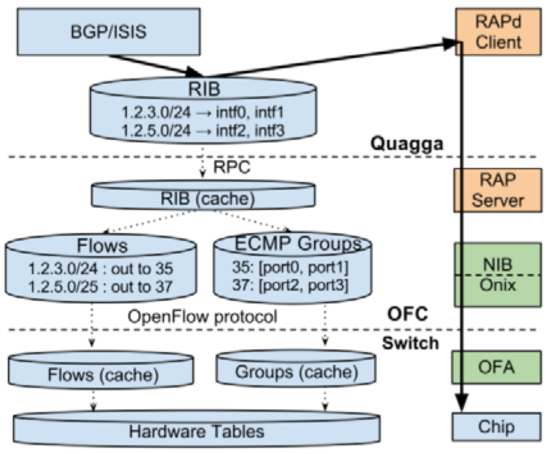

Fig.5 Integrating Routing with Openflow Control[2]

(2)Packet forwarding: As shown in Fig. 6, switches play three roles. The Encap Switch encapsulates packets with a fixed source IP address and a per-tunnel destination IP address based on a hash of the packet header. Flows with different destination addresses are forwarded along different paths. The Decap Switch recognizes these packets that need to be decapsulated, decapsulates them, and then forwards them to the destination site based on the inner packet header.

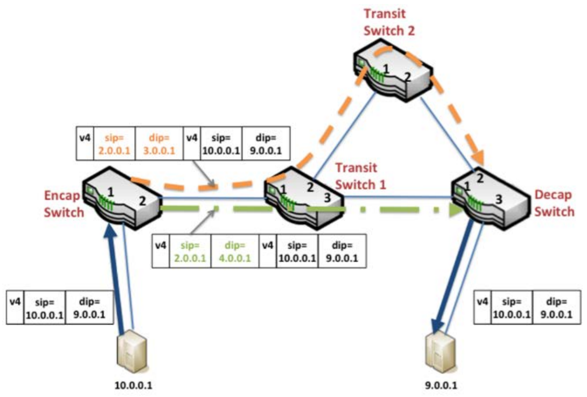
Transit Switch 2
Transit Switch 1
Encap Switch
Decap Switch
v4 sip= 2.0.0.1 dip= 3.0.0.1 v4 sip= 10.0.0.1 dip= 9.0.0.1
v4 sip= 2.0.0.1 dip= 4.0.0.1 v4 sip= 10.0.0.1 dip= 9.0.0.1
v4 sip= 10.0.0.1 dip= 9.0.0.1
v4 sip= 10.0.0.1 dip= 9.0.0.1
10.0.0.1
9.0.0.1

Fig.6 Multipath Wan Forwarding[2]

3.4 Impact of Failures

Table 1 summarizes the result of failure. A single link or an encap switch failure leads to traffic loss for a short time, since the affected switches can immediately find other switches by multipath WAN forwarding. On the contrary, the failure of a transit switch adjacent to an encap switch requires up to 3300 milliseconds. Because these transit switches affect the multipath table entries in encap switches, the encap switches must update table before forwarding. What's more, sometimes, the update involves site controller, which is time-consuming on communication.

The most interest result of failure is OFC and TE server failure, which are all lossless. Thanks to deploying routing and traffic engineering as independent services, failure of one service will not affect the other. For example, if TE server is disabled, the network can run successfully with OFC server.

Table 1 Traffic Loss Time on Failures[2]

Failure Type	Packet Loss(ms)
Single link	4
Encap switch	10
Transit switch neighboring an encap switch	3300
OFC	0
TE server	0
TE Disable/Enable	0

4. TRAFFIC ENGINEERING IN SDN

4.1 Outline

There are two main components in SDN: (1) SDN Controller (SDN-C), which computes and controls in centralized forwarding table for every traffic flow in networks as the topology information, link status and traffic flow information of whole network; (2) SDN Forwarding Element (SDN-FE), which executes forwarding packets based on the forwarding table given by SDN-C, and splits traffic across multiple hops to a destination, and does some measurements for traffic across multiple hops to a destination.

The interactions between an SDN-C and an SDN-FE when a new traffic flow is initialized are: (1) the first packet of flow is sent from an SDN-FE to the SDN-C; (2) SDN-C computes forwarding path for this traffic flow, and updates forwarding tables of all SDN-FEs which are located on the flow path from source to destination; (3) SDN-FE forwards subsequent packets for this flow and no more actions by SDN-C are needed.

In practice the most probable scenario is that a limited number of SDN-FEs is introduced into an existing network, in other word, SDN-FEs and traditional routers co-exist in one network. Therefore, two inevitable issues are: (1) how the SDN-C computes the forwarding paths for the flows according to the traffic flow information provided by the SDN-FEs as well as the traditional routers, in order to achieve higher network performance, and (2) how to locate the limited numbers of SDN-FEs, so that they are put at the best locations in the network.

4.2 Mathematical Modeling

4.2.1 Computing Fowarding Paths

a)Objective: Minimize the maximum utilization of any link in network.

b)Condition 1: Total traffic on the link < maximum link utilization · capacity: The total traffic flow comprises of two parts. The first part is traffic that goes from source to destination without passing an SDN-FE, which is dentoed by $FLOW_{ospf}$. The other part is the traffic that passes through at least one SDN-FE before reaching its destination, denoted by $FLOW_{sdn}$. The first part can be computed according to messages sent by OSPF-TE, while the second part is calculated by information generated from SDN-FEs.

c)Condition 2: The total traffic flow loaded by SDN-FEs can be routed in the network : The total loaded traffic by SDN-FE can be computed by SDN-C upon the measurements done by the SDN-FEs. $FLOW_{sdni}$ is used to denote it, and $FLOW_{ud}$ is used to denote the traffic injected by SDN-FE to the destination.

d)Condition 3: the traffic flow on any path is positive.

e)Solution: To minimize the maximum utilization of any link means to minimize the traffic delay and packet loss, because the link utilization increases as growing of the time delay and packet loss. Minimizing time delay and packet loss is the objective. Ul_{max} is used to denote maximum link utilization, and C_l to denote link capacity.

SDN-C can thus compute the next hops for all injected traffic at the SDN-FEs for each destination. The formula is as follows:

Minimize Ul_{max}

Subject to

$FLOW_{ospf} + FLOW_{sdn} \leqslant Ul_{max} \cdot C_l$

And

$FLOW_{sdni} \geqslant FLOW_{ud}$

An algorithm named FPTAS may be used to solve this problem.

4.2.2 Positioning Sdn-Fes

a)Objective: Maximize throughput.

b)Conditions: (1) The flow in path set P controlled by SDN-C is less than the link capacity; (2) The flow from source to destination along admissible path is greater than the product of throughput and known traffic bandwidth from source to destination.

c)Solution: A primal-dual algorithm may be used to solve this problem. Based on the achievements mentioned above traffic engineering in SDN may be realized as follows: (1) to place the limited SDN-FEs in the existing network reasonably according to the topology, traffic flow of the network; (2) under the control of SDN-C, to improve entire network throughput by dynamic routing scheme while making less delay and loss in the network as the changing traffic matrix.

4.3 Experimental Results

To verify the effectiveness of the mathematical models and corresponding algorithms, experiments and simulations have been done, and the results show: (1) the throughput increases greatly with the increase of SDN-FE nodes, because SDN scheme as well as FPTAS is used; (2) the throughput of the network using SDN routing is much better than that using OSPF; (3) the traffic delay and loss in using SDN routing are much less than that using OSPF.

4.4 Summary

The deployment scene of more and more SDN co-existing with tradition routers or switches will last for a period. It has been proved that improved network performance can be achieved by using SDN incrementally in existing network even a limited number of SDN-FEs are located. The important point is that introducing SDN into existing network has no influence on the routing protocol for traditional nodes using OSPF forwarding.

The formulas and relevant algorithms on using SDN for dynamic routing in existing network have been proved that they can greatly improve overall network throughput while maintaining less delay and loss in the network. SDN scheme is effective, and corresponding algorithms are more important on segment routing.

5. P4: AN EXPLORATION INTO THE POSSIBILITY OF FORWARDING PLANE PROGRAMMING

P4 is a high-level programming language for switches. It introduces openness to the data plane, offering customization of internal details of how silicon processes a packet in a unified way so that the chip vendors' IP will not be jeopardized. P4 was prompted as a solution to new thirst appearing along the development of the OpenFlow system.

5.1 Industrial Background and Motivation

The OpenFlow system was initially designed for experimental purposes that one could devise and trial a new protocol without interfering with production traffic within a small network (e.g. campus Ethernet) [3]. However, as the OpenFlow system has been used as a primary approach to SDN and has been deployed in larger scale of networks[2], and due to the proliferation of new header fields (see TABLE II) and rule tables, OpenFlow 1.x and chips dedicated for OpenFlow must be regularly extended to be up-to-date[3]. Such rapid updating is not preferred when there could be an option to make things flexibly extensible, therefore P4 has been proposed as a new method to define the behavior of switches.

The OpenFlow system was born at the time when programmable ASICs ran 10 to 100 times slower than fixed-function ASICs, and hence Mckeown's team assumed that it'd be run on switches supporting a fixed set of protocols, which were decided by the datasheet of the switch ASICs[4]. What OpenFlow did was merely making forwarding table automatically configurable by a programmable controller. As shown in Fig. 7, classic OpenFlow system only translates and installs rules.

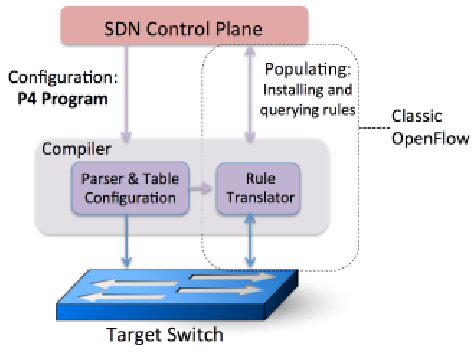

Fig.7 Comparing P4 with Openflow[3]

As the growing need to extend supported header fields and the fact that the performance of programmable switch chips catching up fixed chips in 2014[4], making the data plane "open" became possible. If the exact pipeline within a switch could be programmed flexibly, SDN would step into a new level: instead of only updating forwarding tables, operators would be able to tell the switch exactly how to process packets by designing workflow. That is, rather than simply installing entries in flow tables, as shown in Fig. 7, the controller can also configure parsers and more.

A programmable chip is independent from protocols because the header matching function could be arbitrarily defined, and thus it is rather considered as a protocol-independent packet processor. To program such chips, the language P4 for Programming Protocol-independent Packet Processor was introduced[3].

5.2 P4 Forwarding Model

Table 2 Fields Recognised By the Openflow Standard [3][5]

OpenFlow Version	Date	Header Fields Supported
OF 1.0	Dec 2009	12
OF 1.1	Feb 2011	15
OF 1.2	Dec 2011	36
OF 1.3	Jun 2012	40
OF 1.4	Oct 2013	41
OF 1.5	Dec 2014	38

P4 overwhelmingly changes the way OpenFlow switches work by introducing the use of programmable switches, therefore it comes with a new abstract model of forwarding and packet processing of its own (see Fig. 8).

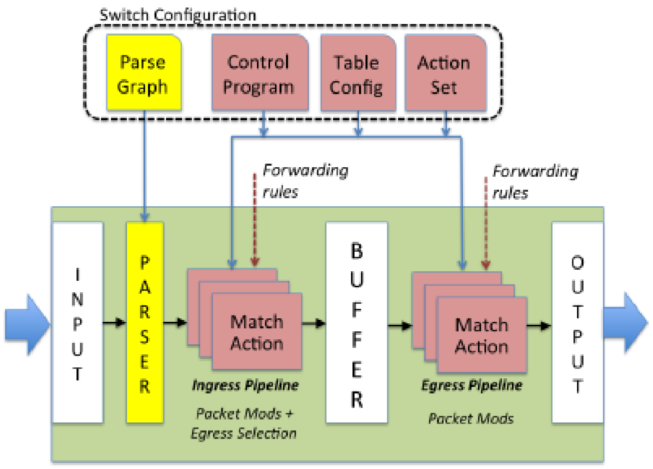

Fig.8 Abstract Forwarding Model of P4[3]

The upper dotted-lined area refers to the configurable elements that exactly define the behavior of data plane.

(1)Parse graph: The Parse Graph defines header patterns to be matched and possible valid sequences of nested headers, on which the parser depends. In P4, the graph is represented by individual header definitions together with definitions of parsers that are linked to indicate the possible series of defined headers.

(2)Control program: The Control Program decides the execution order of tables of match-plus-action (in parallel, in a serial manner or hybrid).

(3)Table configuration: The Table Configuration defines the header fields patterns to match and corresponding actions.

(4)Action set: Programmable switches only offer basic operations shipped with hardware. Any combinations of these primitive actions can be defined as an Action, which in turn can be triggered in a match-plus-action table.

These configurations are written in P4 and can be compiled with a P4 compiler to be installed on programmable switches.

This model is an abstraction of the processing pipeline, independent of the implementation of the switch (e.g. software-based or ASIC-based switch) executing it, and thus designing P4 by following such model allows P4 work on any type of switch implements the model.

When a packet is received, it's first parsed by a programmable parser (adapting new headers) and then passed to sets of sequential or parallel match-plus-action pipelines, and finally forwarded out. A packet's body is irrelevant to the processing [3] and hence only the header fields get analyzed, leaving the body data buffered separately. After reading all header fields, the switch should find the right protocol program to handle this packet by passing it to the ingress matching and action pipeline. Then it's the ingress pipeline's job to determine the action on this packet: forwarding to a certain egress pipeline queue, dropping the packet, making duplicates for further actions (e.g. sending to control plane for table updating) or invoking subprograms. Egress processing takes in packets from its queues. It handles these packets (possibly duplicates of the same packet) individually by matching and action.

5.3 P4 Impacts on Sdn

As a provisional approach to meet emerging needs for data plane programming, P4 has been a project of Open Networking Foundation since 2018[6]. Due to that P4 makes data plane programmable, multiple researchers used P4 to realize their experiment designs[7]. P4-based SDN is also recently used as the backing of new implementation of applications[8].

6. Conclusion

SDN implementations have been widely deployed since SDN was conceived. Despite of multiple imperfections, it still has shown sufficient usage in WAN structuring, traffic engineering, etc. Efforts have also been made on making SDN catching up with industrial demands.

Overall, SDN should be concluded as a visionary idea, considering it has only seen this world for 12 years and yet has grown big enough to attract the industry's attention. Although SDN is still a developing immature network structure, the centralised manner of network designing has been refreshing. With supportive communities and industrial developers’ consistent contribution, it's worthy to believe SDN will embrace a bright future.

Acknowledgment

We thank Dr. William Nace(Carnegie Mellon University) for providing important advice on this paper.

References

[1] N. McKeown et al (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69-74.
[2] S. Jain et al (2013). B4: experience with a globally-deployed software defined WAN. ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 3-14.
[3] P. Bosshart et al (2014). P4: programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 87-95.
[4] N. McKeown, J. Rexford (2016). Clarifying the differences between P4 and OpenFlow.
[5] Open Networking Foundation (2014). OpenFlow switch specification", Opennetworking.org.
[6] P4 Language Consortium (2018). P4 gains broad adoption, foins Open Networking Foundation (ONF) and Linux Foundation (LF) to accelerate next phase of growth and innovation.
[7] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford (2016). HULA: scalable load balancing using programmable data planes", Proceedings of the Symposium on SDN Research - SOSR '16.
[8] F. Hauser, M. Schmidt, M. Haberle, M. Menth (2020). P4-MACsec: dynamic topology monitoring and data layer protection with MACsec in P4-based SDN. IEEE Access, vol. 8, pp. 58845-58858