International Journal of Advanced Science and Technology
Vol. 29, No. 8 (2020), pp. 240-248

Data Plane Programmability: Enabling the Vision of Programmable Networks

B. Amutha[1], R. Jeya[2], Harsh Gondaliya[3]

[1,2,3]*Department of Computer Science & Engineering, SRM Institute of Science and Technology, Chennai, India*
[1]*amuthab@srmist.edu.in,* [2]*jeyar@srmist.edu.in,* [3]*harshgondaliya99@gmail.com*

Abstract

In the past 25 years, networking community has brought a series of developments—Active Networking in the mid-1990s, Software-Defined Networking (SDN) during the period 2007-10, Data Plane Programmability during the period 2013-15, etc.— to create fully programmable networks. A fully programmable network allows network operators to quickly deploy new services into their network and enables networking researchers to easily experiment their research ideas in real networks. Such flexibility accelerates the pace of innovation in networks, benefiting everyone who is dependent on the Internet. In this paper, we provide a study of three major networking paradigms: Traditional Networking, Software-Defined Networking, and Software-Defined Networking with a Programmable Data Plane. We highlight major pitfalls that led to a transition from one networking paradigm to the next one. Finally, we show how programmable data planes enable us to achieve our vision of creating fully programmable networks.

Index Terms: *SDN, Data Plane Programmability*

1. Introduction

During the early to mid-1990s, there was a massive boom in the adoption of the Internet. Since then, there has been a constant need to rapidly innovate in networks. Innovation in networks serves different meanings to different stakeholders. For network operators, innovation means being able to quickly deploy new services into their networks to gain a competitive advantage. For networking researchers, innovation means be- ing able to experiment their research ideas in real networks to solve extant problems. But, it turns out that the rigid nature of our conventional networks severely impedes the pace of innovation in networks.

When an enterprise demands for a new networking feature to its network equipment provider, it takes only a few weeks to write software code for the new feature. But, it takes about 4 years to design and fabricate a market-ready ASIC (hard- ware) that renders the new feature [4]. In our conventional networks, since the network software (control plane that takes the forwarding decisions) is tightly coupled with the network hardware (data plane that does packet forwarding), innovation takes place at the pace of hardware evolution (i.e. about 4 years to introduce a single new feature). Such inflexibility forces network operators to opt for complex workarounds and insert middle-boxes [6] to satisfy their network needs. Hence, there has been a pressing need to make networks programmable, i.e. a network must be flexible enough so that we can change it's behavior through our code.

The following developments have been made to achieve programmable networks and to enable rapid innovation in networks:

a) Active Networking (1990s)*:* It proposed two approaches to achieve network programmability: (a) Capsules Approach: Every packet carries a program that can be executed by a switch. (b) Programmable Switches: A network operator pro- grams switches with the desired packet processing behavior.

ISSN: 2005-4238 IJAST

b) Tempest Architecture: Switchlets (1998)*:* It introduced the concept of network virtualization. Here, multiple virtual switches (switchlets) [9] are run on a single physical ATM switch. A divider partitions switch resources like bandwidth, port space, buffers, etc. across multiple controllers which in turn control each switchlet.

c) ForCES: Forwarding and Control Element Separation (2003)*:* Network Elements (NE) are logically separated into several Control Elements (CE) and Forwarding Elements (FE) [13]. Control Elements control Forwarding Elements through a dedicated channel called the ForCES interface. Control Elements provide routing and signaling functionality while the Forwarding Elements provide general packet forwarding and packet handling functionality. However, any changes to pre- defined protocols need to undergo the IETF standardization procedure and require a change in current hardware. This was the major limitation of this approach.

d) RCP: Routing Control Platform (2004)*:* In this ap- proach, a centralized control platform called RCP [5] computes routes and sends them to the forwarding devices via standard Border Gateway Protocol (BGP). Using BGP as the control channel allows us to make changes in our networks with- out having to undergo any IETF standardization procedures. However, one can only make changes that BGP supports. It doesn't allow us to make major changes to the existing network protocols.

e) The 4D Network Architecture (2005)*:* This architecture [12] consists of four planes: (a) Decision: takes network routing and management decisions (b) Dissemination: used as a channel for communication between the decision plane and data plane (c) Discovery: provides network-wide view to the decision plane and monitors traffic (d) Data: handles network traffic.

f) VINI: Virtual Network Infrastructure (2006)*:* It aimed at providing researchers a platform where they can run live experiments at scale [10]. It consists of four main components:

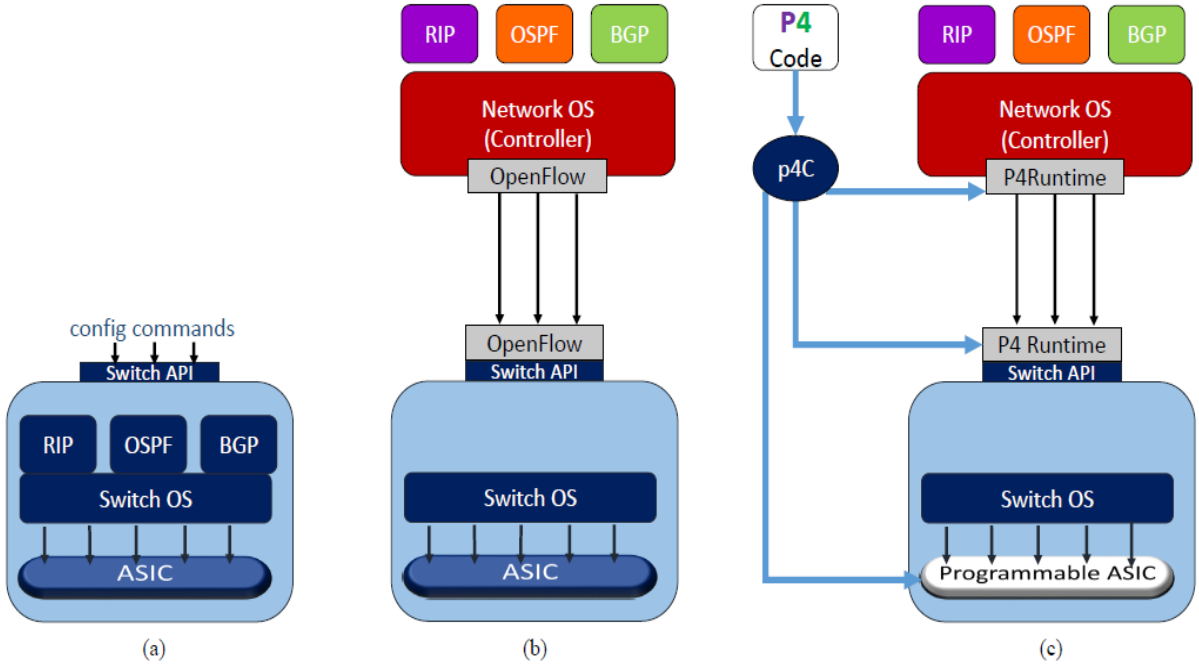

FIGURE 1. Networking Paradigms: (a) Traditional Networking (b) Software-Defined Networking (c) Software-Defined Networking with a Programmable Data Plane

ISSN: 2005-4238 IJAST

(a) XORP: Control Plane that executes routing protocols on virtual network topologies (b) Click: A software router that acts as the data plane (c) Tunnel: Used to implement virtual interfaces (d) Filters: Used to create link failure condition in our virtual network.

g) Cabo: Concurrent Architectures are Better than One (2007)*:* It proposed that the infrastructure providers can operate independently from the service providers [11]. In this way, both will be able to do justice to their expected services and eventually the pace of innovation in networks will also increase.

h) Ethane (2007)*:* This was the first approach that pro- posed customization in hardware to allow network pro-grammability [8]. Here, a domain controller computes flow table entries based on the high-level network policies. It pushes flow table entries into the customized forwarding devices. The only limitation of this approach is that we need to be dependent on such customized forwarding devices that can support the Ethane protocol.

i) OpenFlow (2008)*:* OpenFlow is a standard protocol that is used to program forwarding devices in a Software-Defined Network [1]. All the network equipment vendors expose the common functionalities of routers and switches through an open interface.

SDN Controller uses this standard, vendor- agnostic interface to push down the flow table entries to forwarding devices. However, OpenFlow does not allow us to deploy services that require defining a new packet header or that require complex packet processing action.

j) RMT: Reconfigurable Match-Action Tables (2013)*:* This model allows us to change the configuration of our data plane in the field without modifying the hardware [2]. Hence, network operators and networking researchers can define new header fields and actions as per their network needs.

k) P4: Programming Protocol Independent Packet Processors (2014)*:* P4 is a protocol-independent, target-independent programming language [3] that can be used to configure the data plane of our Software-Defined Network. It allows us to define custom headers and match-action tables. Moreover, un-like many low-level Hardware Description Languages (HDLs), P4 is very easy to learn.

The outline of the paper is as follows. First, we explain the traditional networking paradigm and its limitations. Next, we describe how Software-Defined Networking paradigm overcomes limitations of the traditional networking paradigm to some extent.

Next, we illustrate how Software-Defined Networking with a programmable data plane allows us to overcome all limitations imposed by the previous networking paradigms. Last, we highlight future research directions towards achieving our vision of fully programmable networks.

2. Traditional Networking

As shown in Figure 1 (a), in the traditional networking paradigm, our Switch OS (Control Plane), Switch ASIC (Data Plane), and Network Protocols/Applications (Management Plane) are tightly coupled with each other in a switch box.

A network operator manually configures this switch box using vendor-specific configuration commands. Switch OS receives these commands through Switch API, understands policies defined by the network protocols, and accordingly loads table entries into the Switch ASIC.

ISSN: 2005-4238 IJAST

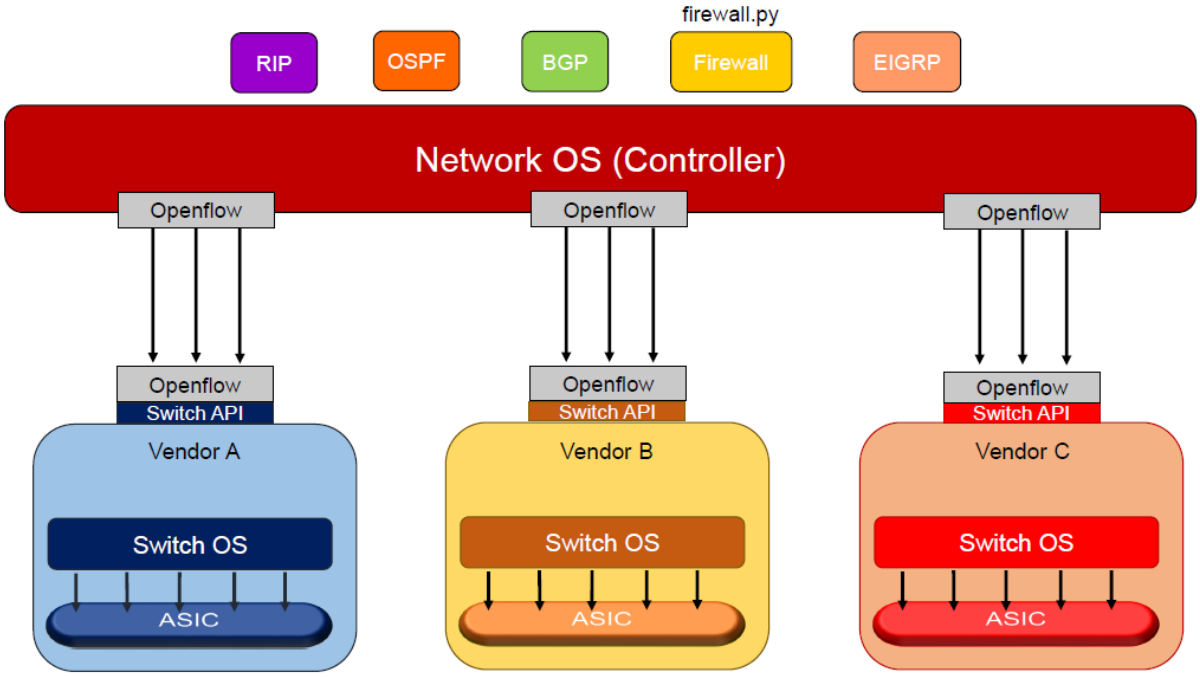

FIGURE 2. Adding a simple new network feature in a Software-Defined Network

A. Limitations

- ***a)*** ***Dependency on middleboxes****:* Traditional Network's rigid structure forces us to use middleboxes to add new network features like NAT, firewall, etc. The addition of such middleboxes increases the complexity of our network and makes it difficult to change our network topology in the future.
- ***b)*** ***Impedes innovation****:* It takes about 4-5 years to intro- duce a new network protocol in the traditional networking paradigm. Consequently, network operators cannot quickly add new features to their networks, and networking researchers cannot easily experiment their research ideas in real networks.
- ***c)*** ***Makes network management difficult****:* Each device in the network needs to be configured manually as per the vendor- specific commands. It makes network management task very difficult and time-consuming.
- ***d)*** ***High capital and operating costs****:* Complex network management and slow innovation leads to an increase in capital and operating expenses of the networks.
- ***e)*** ***Complex distributed state management****:* Traditional net- works use various distributed routing algorithms like Routing Information Protocol (RIP), Distance Vector Routing (DVR), etc. It is challenging to maintain a consistent state and quickly converge a network in case of failure.

B. Scenario: Adding a new network feature

Let us consider a scenario where a network operator wants to add a new feature called "firewall" into their network. In the traditional networking paradigm, it is not possible to quickly add this new feature. It is because: (1) We need to standardize this new feature as a protocol that all network equipment vendors must follow. (2) A change in existing Switch ASIC is required to introduce this new feature. The entire process of fabricating new Switch ASICs and bringing it to the market takes about 4-5 years. Hence, it is not possible to quickly add new features in the traditional networking paradigm. Consequently, we cannot achieve rapid innovation in networks.

ISSN: 2005-4238 IJAST

3. Software-Defined Networking

As shown in Figure 1 (b), in the Software-Defined Net- working paradigm, the Control Plane (Network OS) that takes major packet forwarding decisions is decoupled from the Data Plane that performs packet forwarding. A network operator or a networking researcher can load network protocols as typical software programs into the Network OS. Further, Network OS uses an open southbound interface called OpenFlow to load flow table entries into the data plane devices. Next, the Switch OS loads these flow table entries into the vendor- specific Switch ASIC.

A. Advantages

a) ***Enables network programmability:*** SDN allows us to define our network behavior through a software program. It gives us the flexibility to quickly reprogram our network behavior as per our needs. Thus, network operators can quickly deploy new services into their networks, and networkingvxlan.py researchers can easily experiment their research ideas in a real network.

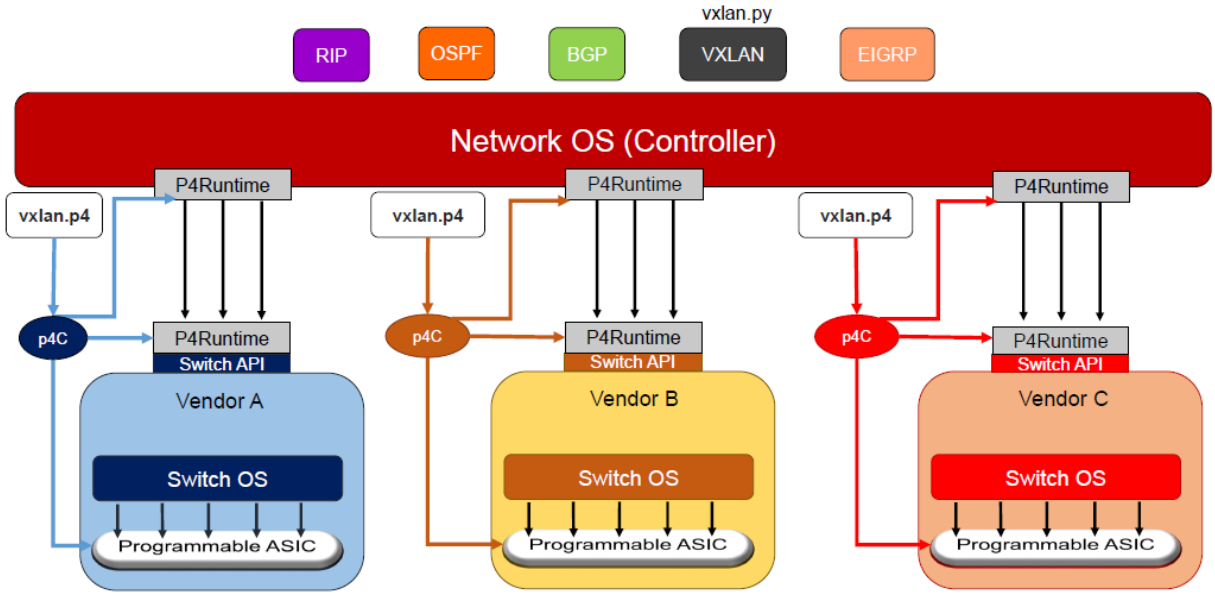

FIGURE 3. Adding a complex new network feature in a Software-Defined Network with a Programmale Data Plane

b) ***Centralized network provisioning:*** SDN provides us a centralized view of our entire network, enabling us to quickly provision network devices in our network.

c) ***Lower capital expenses:*** A network operator can repro- gram existing OpenFlow enabled hardware to incorporate new network features. This eliminates the need to replace hardware while making changes to our network. Hence, it increases the lifetime of our network hardware, reducing our capital expenses.

d) ***Lower operating costs:*** SDN automates the majority of network administration tasks, which saves a lot of operating expenses for the network operators.

e) ***Efficient content delivery:*** SDN provides high network responsiveness and delivers optimum Quality of Services (QoS) for voice, video, and other multimedia applications. This helps provide a satisfactory user experience to the customers.

f) ***Better management of cloud resources:*** SDN allows us to manage cloud resources of a data

ISSN: 2005-4238 IJAST

center from a central platform. Thus, we can actively allocate-deallocate resources as per our needs.

g) ***Effective network security****:* SDN gives us network-wide visibility from a single central point. This allows us to quickly detect a security threat or vulnerability and respond to it.

B. Scenario 1: Adding a simple new network feature

Let us consider a scenario where a network operator wants to add a simple new feature called ”firewall” into his network. Now, in the software-defined networking paradigm, each vendor exposes a basic set of headers and actions defined in the OpenFlow specification. As shown in Figure 2, a Network OS controls multi-vendor data plane devices through OpenFlow API. So, the network operator just needs to write a control plane application *firewall.py* that can provide the firewall functionality. He can load this application onto the Network OS, and the Network OS will push down appropriate flow table entries into the data plane devices. Thus, the network operator can quickly add a new feature to his network without needing to replace his network hardware. Moreover, all the data plane devices, regardless of their vendor, will execute this new firewall functionality. Several such control plane applications have been written to implement various traffic engineering and network security features [14,15,16].

C. Scenario 2: Adding a complex new network feature

Let us consider a scenario of the year 2009, where a network operator wants to add a complex new feature called ”VXLAN” to his network. This feature requires defining a new packet header and packet encapsulation action. It turns out that the OpenFlow 1.0 specification allows us to use only a fixed set of headers and simple packet processing actions to write our SDN control plane applications. To add the ”VXLAN” feature, we need to: (1) define its header in the OpenFlow specification, (2) replace old ASIC with a new one that can openly expose VXLAN functionality. This entire process of changing hardware will take about 4-5 years. So, we see that whenever we want to add a new functionality that requires defining a new header or action that is not available in the OpenFlow specification, we need to change hardware. This shows that the rigid nature of our data plane limits us from achieving fully programmable networks.

4. Software-Defined Networking with a Programmable Data Plane

The fixed-function hardware used in SDN was the major impediment to achieving a fully programmable network. Pro-grammable hardware costs more, consumes more power, and operates at data rates lower than fixed-function hardware. Two developments in the year 2013 have enabled us to achieve our vision of fully programmable networks.

First, the development of RMT (reconfigurable match-action table) [2] switching chips enabled us to have such pro-grammable hardware that costs the same as fixed-function hardware and consumes the same power. Moreover, pro-grammable ASICs like Tofino can function at data rates higher than the normal fixed-function hardware.

Second, the emergence of P4 (Programming Protocol Independent Packet Processors) [3] programming language allowed networking researchers and network operators to easily write software code to program the data plane devices. Unlike the low-level hardware description languages like Verilog, P4 is a high-level language and is very easy to learn. Moreover, P4 is a target and protocol independent language, enabling us to implement new network features and protocols without any restrictions.

So, now when software-defined networking uses a pro-grammable data plane, the network operator needs to program his network in two steps:

ISSN: 2005-4238 IJAST

(1) A P4 program is compiled using a p4c compiler provided by the network equipment provider. p4c compiler configures the SDN data plane with headers and match-action tables defined in the P4 program.

(2) A control plane application is loaded onto the Network OS. Network OS pushes down appropriate flow table entries into the data plane devices through P4Runtime API. Further, the Switch OS loads these entries into the programmable ASIC.

A. Advantages of P4 Language

a) ***Target independent:*** P4 is not tied to any particular vendor-specific hardware target. An equipment vendor pro- vides developers with a compiler and an architecture model. The developer focuses on writing the p4 program and the P4 compiler takes care of compiling it to the target hardware.

b) ***Protocol independent:*** P4 language is independent of the protocols used in the data plane devices. It provides us the flexibility to define our own headers and match-action tables.

c) ***Reconfigurability:*** P4 allows us to change the configuration of our network data plane on-the-fly without having to change the hardware. We just need to reprogram our data plane with a new P4 program.

d) ***Optimum use of hardware resources:*** With P4, we can easily add new features and protocols to our switches. Hence, we can only have the required network protocols and features in our switch boxes. In the future, we can add features as we need.

e) ***Greater network visibility:*** P4 allows us to implement network monitoring applications like In-band Network Telemetry (INT) that can collect telemetry information from switches in the network. This allows us to effectively troubleshoot our networks.

f) ***Full flexibility:*** P4 allows us to define our custom headers, parser block, match-action table, and deparser. Thus, we have the flexibility to implement any protocol that we like without facing any limitation.

g) ***Fosters innovation:*** With P4, network operators can quickly deploy new features to their networks. Network features can be deployed at the pace of software development cycles. On the other hand, networking researchers now have the flexibility to use this high-level language to experiment their research ideas in a real network. Overall, all of these factors lead to an increase in the pace of innovation in networks.

h) ***Easy to learn:*** P4 is a high-level language and a normal user can get hold over it within a week. Also, well-designed tutorials are available to help beginners get started with this programming language.

i) ***Strong community support:*** P4 Language Consortium is a member of the Open Networking Foundation and thus, it receives wide community support. Also, P4 working group members and expert developers actively help beginners over mailing list.

B. Advantages of P4Runtime API

a) ***Target Independent:*** P4Runtime allows us to control data plane devices from different vendors. P4Runtime API is generated from our P4 code itself. Hence, programmers just need to focus on writing P4 programs; the remaining process of generating P4Runtime API, and configuring the data plane is taken care of by the P4 compiler.

b) ***Protocol Independent:*** P4Runtime is not tied up with specific protocols. We can define our own parser, headers, match-action table, and deparser. This allows us to easily add new protocols into the switches without having to undergo any standardization procedures.

c) ***Easily Extensible:*** P4Runtime allows us to extend the current data plane configuration without needing to restart the control plane. When a new feature needs to be added, we just need to load a new P4 program to our P4 compiler, and the compiler will generate new P4Runtime API. Thus,

ISSN: 2005-4238 IJAST

we do not face any major network downtime when such changes are made to our network.

C. Scenario: Adding a complex new network feature

Now, after the emergence of P4 and Data Plane programmability, let us revisit our scenario of adding the VXLAN feature to our network. Our network operator will first write a *vxlan.p4* program and compile it using a vendor-provided P4 compiler. This P4 compiler will: (1) configure our network data plane with a suitable parser, match-action table, and deparser; (2) generate P4Runtime API as per our P4 program. Next, as shown in Figure 3, the network operator will write a control plane application *vxlan.py* that can program the control plane of our network. Hence, the control plane will generate flow table entries as per the policies defined in the *vxlan.py* file and push down these table entries into the data plane devices through P4Runtime API. Next, the vendor-specific Switch OS will populate table entries into the programmable ASIC. So, now we see that our entire Software-Defined Network is fully flexible and we can easily add complex new features into our network.

5. Future Research Directions

P4Runtime API support: At present, very few SDN Controllers support P4Runtime API. Only when the current SDN controllers start supporting P4Runtime API, we can leverage full benefits out of a programmable data plane. Also, data plane devices with programmable data planes need to support P4Runtime API. Only then, a network operator can program data plane device using P4 and use Network OS to load flow table entries.

Formal network verification: Networking community needs software tools that can syntactically and semantically verify the working of a control program and a P4 program. These tools should not only check for syntax errors in network code but also verify whether the desired run-time functionality is being achieved when the code is deployed in a real network. Various simulations can be run to verify whether desired policies and packet processing behavior is being achieved.

P4 support on multiple platforms: Slowly, P4 is gaining traction and a couple of vendors have already started providing switching chips that can be programmed using P4. Moving on, all major networking equipment providers need to start supporting P4 for programming their devices. Once this is achieved, we can have fully programmable networks.

New data plane abstractions: With data plane programmability, we can devise new abstractions that allow us to perform sophisticated packet processing operations like dynamic load balancing, source routing, encrpytion-decryption, etc. at the data plane itself. Such abstractions will also improve performance and security of our data plane devices.

Automating P4 code generation: Research efforts can be put towards developing tools that can automatically generate P4 code as per the high-level control programs. Thus, a network operator just needs to write a control plane application, and the Compiler or Network OS will generate a P4 program with suitable parser, match-action table, and deparser definition.

6. Conclusion

This paper provides a survey of the three major network- ing paradigms: traditional, software-defined networking, and software-defined networking with a programmable data plane. Advantages and Limitations of each paradigm are explained with the help of a real-world scenario. Furthermore, we have illustrated how data plane programmability allows us to achieve our long-standing vision of having fully programmable networks. Last, future research directions and open issues are highlighted.

ISSN: 2005-4238 IJAST

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, *OpenFlow: enabling innovation in campus networks*, ACM SIGCOMM Computer Communication Review, 2008.
[2] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. 2013. *Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN*. In Proceedings of SIGCOMM.
[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. 2014. *P4: Programming protocol-independent packet processors.* SIGCOMM CCR 44, 3 (2014), 87–95.
[4] *Why Does the Internet Need a Programmable Forwarding Plane with Nick McKeown.* URL: https://www.youtube.com/watch?v=zR88Nlg3n3g (Accessed: March 2019)
[5] N. Feamster, H. Balakrishna, J. Rexford, et. al., *The Case for Separating Routing from Routers*, Proceedings of the SIGCOMM FDNA, 2004.
[6] Kreutz, FMV. Ramos, PE. Verissimo, CE. Rothenberg, et. al., *Software-defined networking: A comprehensive survey*, Proceedings of the IEEE, 2014.
[7] N. Feamster, J. Rexford, E. Zegura, *The road to SDN: an intellectual history of programmable networks*, ACM SIGCOMM Computer Com- munication Review, 2014.
[8] M. Casado, MJ. Freedman, J. Pettit, J. Luo, N. McKeown, et. al., *Ethane: Taking control of the enterprise*, ACM SIGCOMM Computer Communication Review, 2007.
[9] J.E. van der Merwe, S. Rooney, L. Leslie, S. Crosby, *The Tempest- a practical framework for network programmability*, Network, IEEE 12.3 (1998): 20-28.
[10] A. Bavier, N. Feamster, et. al., *In VINI veritas: realistic and controlled network experimentation*, ACM SIGCOMM Computer Communication Review, Vol. 36. No. 4. ACM 2006.
[11] N. Feamster, L. Gao, J. Rexford, *How to lease the internet in your spare time.*, ACM SIGCOMM Computer Communication Review 37.1 (2007): 61-64.
[12] A. Greenberg, G. Hjalmtysson, et. al., *A clean slate 4D approach to network control and management*, ACM SIGCOMM Computer Com- munication Review, Oct 2005.
[13] L. Yang, R. Dantu, T. Anderson, R. Gopal, *Forwarding and Control Element Separation (ForCES) Framework*, RFC 3746, April 2004.
[14] A Pillai, MS Vasanthi, R Kadikar, B. Amutha, *Encryption analysis of AES-Cipher Block Chaining performance in Crypto-Wall Ransomware and SDN based mitigation*, International Journal of Engineering Tech- nology, 7 (2.24), 47-54, 2018.
[15] VD Chakravarthy, B. Amutha, *Path based load balancing for data center networks using SDN*, International Journal of Communication Systems, e4213, 2019.
[16] A Phatak, R Kadikar, K Vijayan, B Amutha, *Performance Analysis of Firewall Based on SDN and OpenFlow*, International Conference on Communication and Signal Processing (ICCSP) 2018.

ISSN: 2005-4238 IJAST