ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

The Overview of SDN Architecture and its Practical Application with Improving Methods

Zhipeng Hui[1], Sijie Zhang[2], Yichen Shao[3], Tongyu Chen[4]

[1]Department of Computer Science & Department of Cognitive Science, Rensselaer Polytechnic Institute, New York 12180, USA
[2]Department of Digital Media Technology, Xiamen University, Xiamen 361005, China
[3]Department of Engineering, University of California Santa Barbara, California 93106, USA
[4]Hangzhou Foreign Language School, Hangzhou 310023, China

Abstract: *This paper presents a comprehensive analysis on Software Defined Network (SDN) based on these following main aspects: i) an overview of the SDN architecture, ii) the introduction of a practical SDN application, iii) characteristics and drawbacks of SDN, and iv) methods for improving the performance of SDN. By reviewing and researching the rise and development of SDN, it is proposed that SDN solves many issues of traditional distributed networks, including configuration and management difficulties due to network node diversity. However, it still has some limitations, such as decentralize tendency and code opacity caused by competition among vendors. The goal of SDN development is to put SDN in the details of real issues, to improve it from different aspects, and thus make it as a better network.*

Keywords: *Software Defined Network, SDN OpenFlow, Traffic Engineering Algorithm, Centralized Traffic Engineering, Limitations*

1. Introduction

The core idea of SDN is to separate hardware from software. Let the hardware to do the simplest job and let the software to be the leader. Hence, the protocols on software have rapid iteration without the limitation or revision of the hardware.

This paper combines the ideas from four research papers of SDN. Everything started with the birth of OpenFlow which is a programmable network model in campus networks [1]. Later, Google deployed B4, a global Software Defined WAN, to connect Google's data center in 2013 [2]. Comparing to the traditional WAN architecture, B4 defines the priority of the applications which enables B4 to drive the link utilization to nearly 100% and has control over the edge servers and the network. Moreover, SDN is the latest interacting model dividing the controlling level of the network from the level that forwards data packets and ensures that it offers unified applications. A central controller that is not physical and has the widespread view of the network. It works to ensure that it monitors the decision-making process, and it connects with global data forwarding elements through standard connectors.

Although SDN has many seemingly attractive advantages and was praised as an ideal goal to reach when in practical use, it does not completely achieve its promises. White et al. gave a detailed analysis of the three promises of SDN-centralization, flow control, and programmability, and proposed the drawback and limitations of SDN in their research paper [3]. Specifically, the first and third features no longer hold as SDN scales and vendors compete, while the second brings about a burden for connection establishment and transfer efficiency. Similar research on the characteristics of SDN includes [4] and [5].

However, because managing networks in tasks such as routing and traffic monitoring is unnecessarily complicated and error-prone, SDN is poised to change this by offering an interface between networking devices and the software that controls them. As a matter of fact, for today's SDN controller, platforms for writing applications remain in low-level distributed programming. As a replacement to the low-level imperative interfaces, Frenetic offers abstractions for querying network state, defining forwarding policies, and updating policies [6]. There are also several tools to manage SDN structures, such as Flowsim, a tool that provides interactive SDN switch visualization [7], and MISSIn system, a supportive tool for orchestrating SDN infrastructures [8], and Mininet, a manageable tool to implement the flow admission control module [9].

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

In order to make SDN more robust and efficient, research on improving SDN structures has been conducted in a variety of aspects. For instance, there is research on enhancing SDN security [10][11], manners to setting multiple controllers [12], and implementing caching policy in SDN controllers [13].

2. Background

Software Defined Networks (SDN) is a technology to improve the network with the separation of data and control planes. There are some essential characteristics of an SDN architecture.

The first is the switch's flow table--this is probably an important task of the SDN control plane to manage and install flow table in all of the switches. In an OpenFlow system, it is possible for controllers to confirm a flow that succeeded in the registration on a flow table. More specifically, a controller transmits and receives an OpenFlow message. The switch is with a flow table which manages flows registered from a controller, and a flow table reading section which reads corresponding data, a flow table control section. The flow table has a normal one and a save one. A valid flow entry is registered on the normal flow table. An invalid flow entry which does not conform to the configuration data of the switch is registered on the save one.

The second is the separation of data plane and control plane. The data plane which contains servers and software consists of the network's switches that execute the "match plus action" rules.

The third is the network control functions—it is not surprising to see SDN isn't quite alike to traditional routers. On the contrary, this software executes on servers that are both distinct. The control plane itself consists of two components--an SDN controller, a set of network-control applications.

3. The Rise of SDN

Networks have been developed over decades. The dilemma networking researchers facing is the implementation of the practical experiment with some new network protocols. However, OpenFlow surprised us with a whole new concept of a programmable networks model in campus networks [1].

Researchers discovered that most modern Ethernet switches and routers contain flow-tables which have a common set of functions running in the flow-tables of switches and routers. Exploiting the common set of functions in flow-tables, OpenFlow introduces a new protocol to program the flow-table. Hence, it provides the possibility to program in different switches and routers.

3.1 Basic Model

The basic model of OpenFlow Switches mainly contains three parts. The first one is the flow table which is defined as all packets matching to a specific header. The second one is the Secure Channel. This one provides the connection between the switch and a remote control process (controller). The last one is the OpenFlow Protocol which enables the controller to communicate with a switch.

The working mechanism of the flow-table entry also contains three fields. For sure, at the matching field, a range of selected packet header defines the flow. At the action field, when an incoming packet matches the matching field, then, actions will be executed and decide how the packets should be processed. The last field is Statistics, or it can be called as Counters. Its job is to count and have a track of the number of incoming packets and bytes for each flow. Because each flow entry associates with a simple action, the "Type 0" OpenFlow Switches defines three basic actions for its work. Firstly, the flow's packets are forwarded to a given port. This action allows incoming packets to be routed properly through the network. Then, the flow's packets are encapsulated and forwarded to a controller as the second action. The packets are delivered to Secure Channel mentioned before. In general, the first packet in a new flow is used to do so. Hence, within the help of Secure Channel, the controller can decide whether this new flow should be added to the Flow Table. In contrast, if the Secure Channel denies the passing of the first packet, the third action will be executed. The new flow's packets are dropped. It helps to restrict the denial of service attacks. Just like the traffic police, it reduces the suspicious broadcast discovery traffic from end-hosts. At last, if the Secure Channel passes the packet, then the flow's packets are forwarded through the pipeline. However, the model of OpenFlow Switches is not simply limited as "Type 0". With the development of networks and the emergence of a set of features, the model can also be defined as "Type 1" switch. The features include rewriting of packet header's portions, arbitrary fields' matching process with the packet header. The features enable researchers to have experiments on new non-IP

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

protocols.

A controller in the Secure Channel adds and removes flow-entries from the Flow Table on behalf of experiments. A controller could also be more sophisticated that dynamically add or remove flows as an experiment's progress. Here could be a new topic of using a neutral network to enable the functionality of dynamically adding and removing flow-entries. On the other hand, the restrictions could also be considered as use of permissions for users. The individual researchers have limited power to control flow entries. The permission-like way controller implemented is the key nature of these mechanisms and the aim of the research. Here is an example of a network of OpenFlow-enabled commercial switches and routers Figure 1 [1].

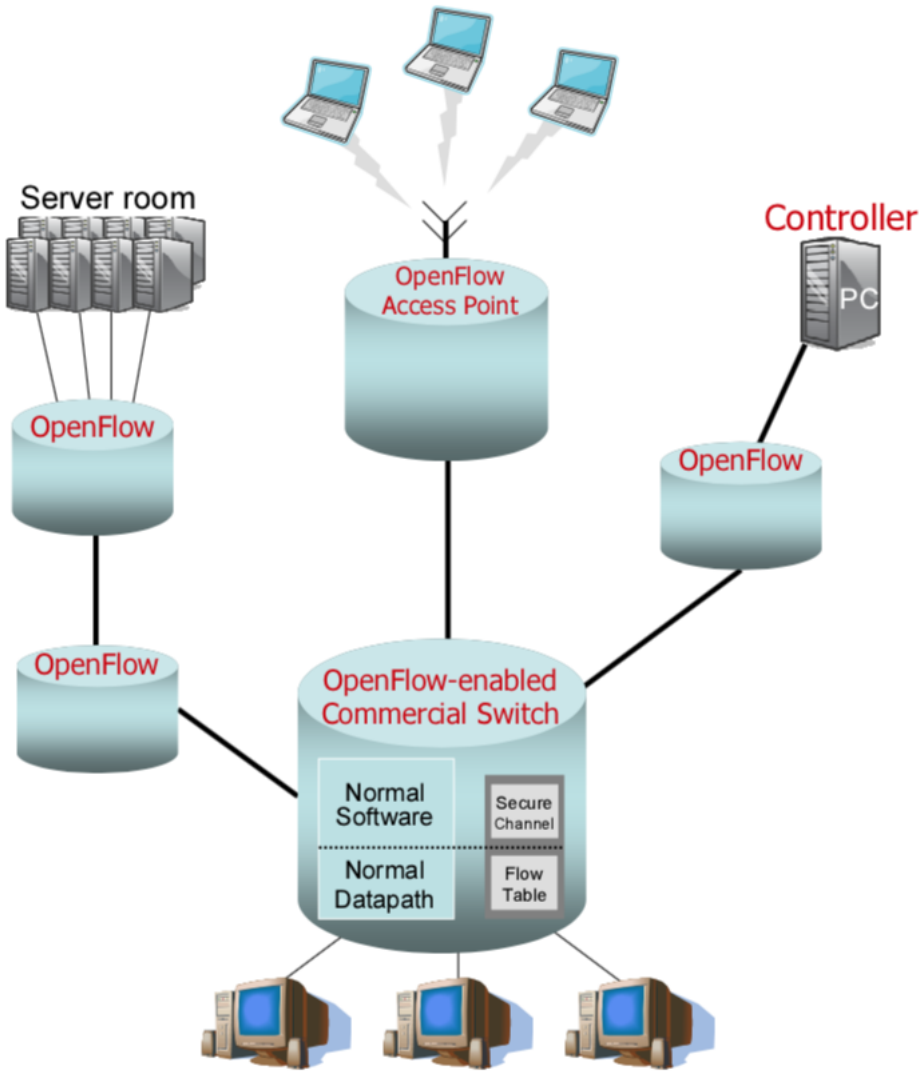

Figure 1. Example of a network of OpenFlow-enabled commercial switches and routers [1]

3.2 Concerns

The model of OpenFlow in the campus network is just a prototype which aims to achieve the goals: 1) Amenable to high-performance and low-cost implementations. 2) Capable of supporting a broad range of research. 3) Assured to isolate experimental traffic from production traffic. The first goal could be achieved by the OpenFlow Switches. The second aim could be achieved by the implementation of the controller. And the third goal could be achieved by the Secure Channel as a whole. However, the paper lacks the discussion of the OpenFlow Switches and security problems. As the OpenFlow Switch contains

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

two parts: software and hardware. That implies that the switch should contain the application layer which is different from normal switches. That could lead to multiple problems. Moreover, as the paper states that a researcher might control the complete network of OpenFlow Switches and be free to decide how flows are processed. That could lead to some serious security problems as there is no super user of the entire network.

4. Google's Deployment of SDN Wan Architecture

4.1 The Reason of B4's Deployment

The Google Company operates on two distinct WANs (wide area networks). The first one is user-facing networks for end-users to deliver their requests and responses to Google's data centers [2]. The second one is the protagonist, B4, in this paper. B4 is used to connect data centers. The birth of B4 helps network companies to save money on internet equipment. The traditional WAN architecture considers all the applications equally. In other words, it treats all bits equally no matter whether or not they deserve. It is known that the packet loss is inevitable in data transmission because a queue preceding link only has a finite capacity. Once it is full, it will drop the new coming packet. At that moment, packet loss occurs. In order to prevent the users from sensing the packet loss, network companies have to buy lots of high-end routing gears to increase the bandwidth. With the help of large bandwidth, even though the packet loss occurs, the network still has spare bandwidth to do the re-transmission quickly. However, the shortcoming is obvious that the average utilization is only 30% ~ 40%. The waste of network resources and expensive equipment leads to the deployment of B4.

4.2 Design Decisions

4.2.1 Separate Hardware from Software

The core idea of SDN is to separate hardware from software. Let the hardware to do the simplest job and let the software to be the leader. Hence, the protocols on software have rapid iteration without the limitation or revision of the hardware.

4.2.2 Drive Links to 100% Utilization

In order to achieve 100% utilization of the network, Google categorizes applications which run on B4 and defines them in different levels and different priority. The volume, latency sensitivity, and overall priority are three key factors to determine the traffic classes. The first one has the lowest volume, highest latency sensitivity, and highest overall priority. Then, the rest is ordered in decreasing volume, decreasing latency sensitivity, and decreasing priority. The top of the pyramid is user data copies, such as email, documents, and video files because users' satisfaction is significant. The users care about the latency and availability most. These things have the lowest volume in general. The second one is remote storage access such as fragmenting data and put them in different data centers or doing the computation on distributed servers. The last one is synchronization. Like if doing the computation on lots of servers in different data centers, servers need to interact with each other and know the states on the other servers.

The idea here is to fulfill the demand of the highest priority mission. And the rest of the bandwidth is filled by other missions which are not urgent. The shortcoming is that packet loss becomes inevitable, or to say that it is more obvious to the servers. As mentioned before, the traditional routing treats all bits same, so everyone has packet loss. However, large bandwidth hides it and make users or servers cannot sense it. The network has free bandwidth to do the retransmission. Even though, suddenly large bandwidth is needed, we can afford it. However, when the utilization reaches nearly 100%, there is no free bandwidth to hide packet loss and no ability to hide the applications with lower priority. The bandwidth is mostly used by higher priority applications. Once the link failures happen, there is no free bandwidth and packet loss cannot be fixed in a short time. Controllers have to allocate the bandwidth dynamically and rob the bandwidth from higher priority applications. It needs time, at this time, the servers or end-users will sense it.

4.2.3 Centralized Traffic Engineering

The first SDN application is called TE, traffic engineering. It mainly runs in the TE server and has three missions. First, TE controls the limited network resources at the edges. It decides who gets the resources when several applications compete for network resources. Second, it uses multi-path tunneling to leverage network capacity. For the same application or flow group (applications with the same priority),

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

they can be assigned into several tunnels to be transmitted. At last, when some links fail to work or shifting application demands, it dynamically reallocates bandwidth. The existence of TE makes sure the network resources are used effectively.

4.2.4 B4 Routers

The traditional view is that the wide-area router must have great buffer capacity. The larger buffer capacity means larger forwarding tables needed for hardware. Forwarding table maps addresses to outgoing link. However, this functionality makes the hardware more expensive and complex. On the other hand, Google finds that most of the switch failures are due to software instead of hardware. Hence, Google chooses to invent its own routers and switches based on SDN architecture and built from merchant switch silicon. The first reason is that the existing routers cannot fulfill the demands of SDN deployment. The new routers enable to separate software from hardware. And also, Google does not have lots of data centers which means they do not need the hardware with large forwarding tables. The second reason is that there are thousands of hardware, and it is impossible to unify all of them in a new language. The new router is able to cooperate with other existing hardware.

4.3 Three-Layers Software Defined Network Wan Architecture

4.3.1 Bottom Layer: Switch Hardware Layer

The bottom layer is switch hardware (Fig. 2 [2]). It only forwards the traffic and has nothing to do with any complicated things like ergates who follow the queen ant's orders and do the hardest job. Here, the OFA (OpenFlow Agent) is a translator and supervisor. Its job is to translate the raw data, the status of the switch, into something OFC (OpenFlow controller) can understand, like whether the switch is busy or forwarding data.

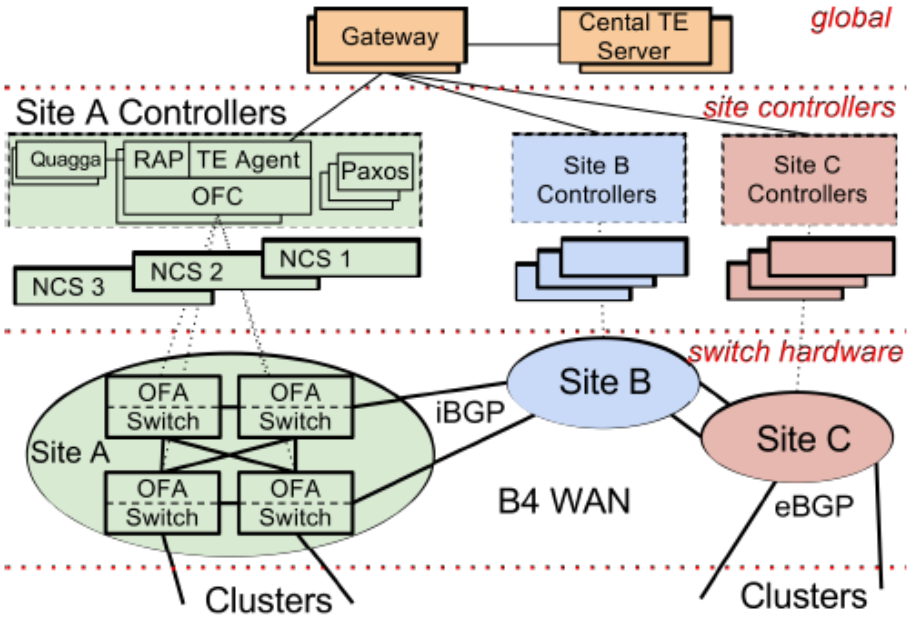

Figure 2. Three layers SDN WAN Architecture [2]

4.3.2 Middle Layer: Site Controllers

The middle layer is site controllers which hold OpenFlow controllers (Fig. 2). Its mission is to decide how to forward the traffic according to switch hardware's status. They are leaders of the departments in the company and give orders to workers. Now, OFC understands working status. So, it communicates with Quagga by RAP. RAP (Routing Application Proxy) is a bridge between them. The Quagga is three layers protocol stack. It contains all of the information about the sites, the paths, the ports. Then, the Quagga provides information back to OFC and OFC uploads all of the information to the TE agent. TE agent uploads to Gateway.

4.3.3 Top Layer: Global Layer

The top layer is a central traffic engineering server which logically controls the entire network (Fig.

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

2). The Gateway is a secretary. It translates the topology, the raw data into something TE can understand and gives these data to the central TE server. The TE server is CEO and does the calculations, like which application go which path. The following part is easy. CEO's decision goes back to OFC and writes in switches. Now, switches know what to do.

4.3.4 Protocols

At the bottom layer, Google chooses to use BGP instead of relying on TE individually to allocate paths and communicate with other routers. The first reason is that there are still existing non-SDN WAN implementation. BGP is required to maintain the functionality of the entire network. Once TE fails to work, BGP can still support the network to work. Second, if a packet is sent from the router, TE is selected as the first choice. However, if one packet passes the router and its information is uploaded to the TE server to calculate the path, this process consumes lots of time. This process is already done on other data centers. Here, the BGP works and helps it to forward the packet directly. However, once some paths have trouble and complicated calculations is needed to figure out the path, TE is used again to deal with it but not BGP.

Paxos at the middle layer is a protocol that works at the application level. It detects failures among the controllers. When some of the controllers fail to work, Paxos elects a new leader from the rest of the controllers. Its job is to make sure the entire network working unerringly when failures occur.

4.3.5 Centralized TE Architecture

In this subsection, it is presented how TE server works (Fig. 3 [2]).

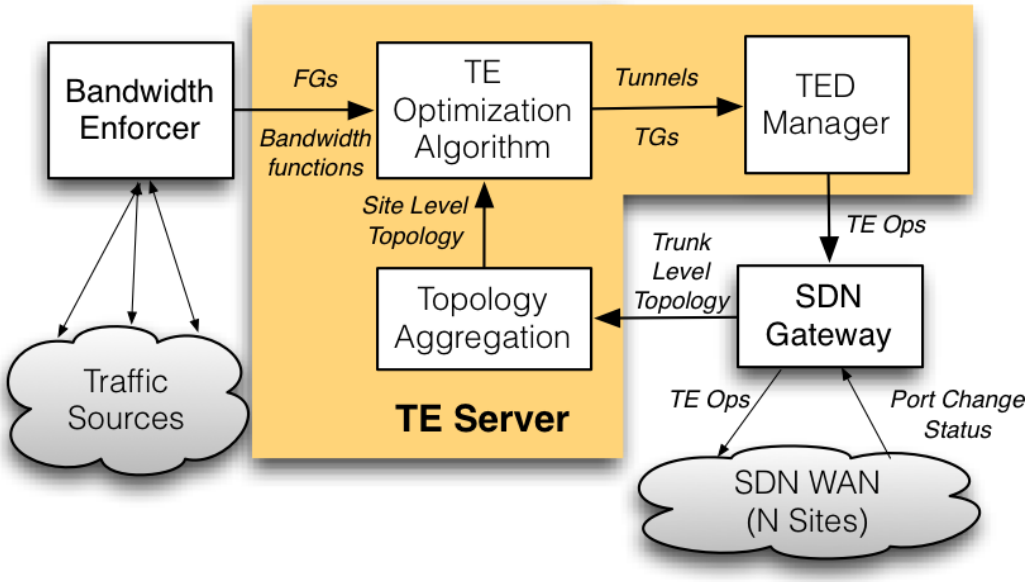

Figure 3. B4 architecture overview [2]

The TE forms the network topology first. The sites are vertices and the lines between vertices are edges. The data is transmitted on these lines. In reality, there are thousands of applications running simultaneously. The process of tracing each application consumes time and adds work to routers. When traffic information is uploaded into the TE server, Google puts similar applications based on priority into a flow group as a unit. The flow group is defined as a tuple. It contains three elements which are source site, destination site, and the QoS (quality of service). When the flow group transmits, its path is called tunnel. For instance, a flow group has three ways to go from site A to site B. It reaches site directly or goes through some other sites. Three tunnels are combined together to form a tunnel group. Here, the TE Optimization Algorithm allocates bandwidth to flow groups based on priority and changes split ratios in each tunnel group. Flow groups are assigned to tunnels, and it is also necessary to consider whether the hardware in the site is able to support forwarding such a number of applications. At last, the Tunnels and Flow Groups are transmitted to the Gateway. The Gateway forwards them to TE agents and OFCs. OFCs do the end job and give orders to switches at the bottom layer to forward them.

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

5. Promises and Limitations of SDN

The concept of SDN was introduced about ten years ago. Representative work includes the previous two paper-OpenFlow and Google B4-and many other researches done on the structure design and model presentation of SDN. For a while, the public had much expectation on this emerging concept. Seeing that SDN has many seemingly attractive advantages, such as centralization and programmability, the business market and media hyped it up, praising it as an ideal structure. However, SDN is not a perfect network as it seems to be. When in practical use, it does not achieve its promises. White et al. gave a detailed analysis on the three promises and characteristics of SDN-centralization, flow control and programmability-and proposed the drawback and limitations of SDN in their research paper [3].

5.1 Centralization

5.1.1 Centralization Structure

One of the biggest differences between SDN and traditional networks is the main structure. Traditional networks, such as IS-IS and BGP, use distributed control planes. The whole network consists of many network nodes, each of which includes a control plane and a data plane. The control plane is responsible for managing the node's operations and communicates with neighbor nodes. Each node can decide its own settings, which is to say it is configured individually.

The drawback of the traditional network is its distribution. Because the configurations of nodes are totally different, the network operator has to store the information for every node and manage complex policies between them. The huge amount of information storage then leads to the adoption of large and expensive devices.

By contrast, SDN uses a centralized structure to eliminate the complexity of the traditional networks. The core component of SDN is the controller. It works as a network operating system, which receives requirements from network applications, calculate the forwarding path according to a set of rules, and send the forwarding command to forwarding devices. Because there are no individual configurations for nodes, and only the controller, but no routers, takes charge of the topology and calculation process, it is easier for the network operator to manage information, and forwarding devices can be cheaper and lighter than the traditional ones.

However, complexity still exists in this situation. In reality, in case the controller is crushed, and also to ensure the SDN service can have global coverage, there has to be multiple controllers to run simultaneously. Therefore, the problem lies in how to deal with their relationship. This topic will be discussed in 5.1.4.

5.1.2 Reactive Forwarding

The traditional networks take a proactive method in the forwarding procedure. That is to say, the forwarding path will be determined before the first packet is transmitted in the network. This method is accomplished by information aggregation of the routers in the network. They hold the accurate information of their neighbors through the process of flooding, processing, and managing, and thus can decide the correct path.

However, for an SDN network, it takes a reactive plan. Forwarding devices can only know reachability information when it receives the first packet. This is because for the controller, due to the large scale and complication of the network it takes control, it does not store all the forwarding information. Therefore, it adopts some protocols and policies to decide possible paths as the forwarding paths of the first packet. If the packet arrives at a forwarding device successfully, the device will feedback this reachability information to the controller.

The issue of reactive forwarding is that, before the forwarding devices receives the first packet and send the information back, the controller does not know the reachability of paths, which causes a disconnection to the real situation. For the same reason, there will be a lag between the first packet beginning to be transmitted and the path to be determined.

5.1.3 Reaction to Changes

For a traditional network, when the state of a node changes, other nodes in the network receive the change information from its neighbor, and send it out to other neighboring nodes until all the nodes are aware of the change, and then the new path will be calculated. Therefore, the reaction time is mainly spent on routers spreading messages and recalculation.

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

When a local node changes or fails in SDN, first the message will be transmitted to the controller. The controller will then calculate and decide a new path and transmit new forwarding rules to related nodes. Although the recalculation is much faster for a controller since it has limited information for nodes, still the spreading of new rules to the forwarding devices in the network is a slow process. Therefore, the reaction time for SDN is no faster than the traditional network.

5.1.4 Reality of Centralization

Centralization, one of SDN's big promises, seems to be a very appealing advantage. Frustrated by a complicated traditional distributed network, SDN's method of centralizing control module will make management a lot easier. Totally physically centralized, however, is impossible in the real world. Firstly, to ensure reliability, there should be multiple controllers working together, while each of them gets a slice of workload. Secondly, to ensure scalability, controllers should be separated into different regions, with each region handled by an SDN system. As a result, SDN still deals with the whole network in distributed systems, which is inconsistent with its promise to be centralized, and the biggest problem emerges: how do different controllers communicate and exchange information. To make the SDN system uniform, people have to decide the format and new protocols applied between controllers.

5.2 Flow-Based Network

The forwarding of SDN is flow-based rather than the destination-based method of a traditional network. It aims to assign different paths in the network for individual applications to ensure that files with special requirements are transmitted efficiently. Flow-based forwarding enables more flexible and individual transmission, but it has significant disadvantages. Firstly, it produces too many connections. For traditional networks, since it only addresses but not files that matter, only one TCP connection is needed between the source and destination, and all files are forwarded through it. However, for SDN, due to the different paths that different files take, it has to set up a TCP connection for each flow. The setup rate will also be slower because the format of the flow should be stored. Secondly, to store a huge amount of connection information, the hardware costs will be quite high. Also, there will be more cost for each forwarding device to examine the headers of flows to ensure they are in correct format and path.

5.3 Programmability

The most appealing part of SDN which attracts the business market is that it promises to be open and programmable. This feature did hold true in the beginning stage, providing a clean and flexible platform for users to develop their own functionalities. However, as SDN becomes more and more widely covered, eventually vendors dominated the market, and they compete with each other. In order to make themselves stand out, vendors develop new models continuously, along with new hardware and software requirements. Meanwhile, they prefer to make their product vertically integrated, which means they tend to develop the whole functionality from application interfaces to controllers, and finally down to forwarding devices in order to make more profit. As a result, the configurations of SDN is likely to be diverse because of different vendors and is no longer uniform as people expect it to be. And on the other hand, vendors are reluctant to disclose their programs, which has led to the development of SDN models being closed within these vendors, and thus is not programmable as it was originally guaranteed.

In conclusion, the concept of SDN appeared bright when it was first proposed, but as it developed, the problems of its characteristics were also exposed. The three main features-centralization, flow control, and programmability-are either flawed or invalid in the long run. Therefore, just as the author said, we should not view SDN as an ideal goal to reach. Instead, it is just a tool or method that needs to be improved.

6. Languages for Software-Defined Networks

6.1 Frenetic

Because managing networks in tasks such as routing and traffic monitoring is unnecessarily complicated and error-prone. SDN is poised to change this by offering an interface between networking devices and the software that controls them.[6]. Although SDN makes it possible to program the networks, it is still inefficient. In order to reach SDN's full potential, there are three designing of simple abstractions for network management. The network traffic is monitored by software to better control. Then, packet-forwarding policies are specified and composed. At last, the policies are updated in a consistent way.

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

Before, on each device, network operators must configure every protocol. However, the growing interests in SDNs nowadays promote a logically centralized controller that manages a distributed collection of switches. Programmers design the packet-forwarding rules and install them on each switch with the help of SDN. This enables programmers to control the behavior of the network directly. Some may regard this as a centralized control, though, for a higher scalability and fault tolerance, the controller is replicated and distributed. SDNs can simplify applications and build up platforms for developing new applications.

Conserving energy: Programmers can use the controllers to shut down links selectively. Sometimes, the whole switches can also be shut down if it is needed.

Balancing the load between servers: When congestion happens, the controller can split flows according to their weights and allocate them to new paths. As a matter of fact, for today's SDN controller, platforms for writing applications remain in low-level distributed programming. As a replacement to the low-level imperative interfaces, Frenetic offers abstractions for querying network state, defining forwarding policies, and updating policies.

6.2 Querying Network State

This is made possible by the underlying run-time system, which follows OpenFlow rules.

Frenetic demonstrate ways of the "control loop" for running a network. One of them is Querying network state. In general, the working state is based on a high-level query language which includes traffic statistics and topology changes.

6.2.1 Query Language Design Considerations

In general, the monitor applications use the packet headers to classify or weight traffic.

For instance, if the programmers define all web server traffic with IP source address 1.2.3.4, there are two rules. Firstly, the upper matches packets source port 80 with TCP source from address 1.2.3.4. Then, the lower one is associate with the rest of traffic. Frenetic specifies predicates like "srcip!=1.2.3.4 & srcport=80". Dynamic unfolding: The usage of efficient hardware and the counters remain the information needed to implement. Limiting traffic: Sending the first packet to the controller and install rules for dealing with future packets. Polling and combining statistics: The run-time system queries the traffic counters and return results to application.

6.2.2 Example Frenetic Queries

MAC learning and Traffic histogram is an Ethernet switch. Its job is to identify what interface to use to reach a host.

```
Select (packets) *
    GroupBy ([srcmac]) *
    SplitWhen ([inport]) *
    Limit (1)
```

The result of this process is a stream of packets to ingress port [6]. Frenetic's language shows the cost of a query to the programmers. The packet-forwarding rules after the first packet for the MAC-learning query are installed gradually. Then, the run-time system directs the packets to the controller.

6.3 Composing Network Policies

Reconfiguring the network: This is a way without having to manually install, actually allows programmers to reconfigure the network. The run-time system is also an insurance to the update process. All of the packets or flows should be processed strictly with the old policy or the new one instead of a mixture. This guarantee ensures that functional invariants are never separated and broken during periods of policies' transition.

6.3.1 Creating Modular Programs

A. Creating Modular Programs

This is a method that considering a program which blend in repeater functionality with web-traffic monitoring functionality.

For instance, the following part is the program that programmers write [6]. Frenetic enables them to

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

compose independent modules directly.

```
def repeater():
    rules=[Rule(inport:1, [fwd(2)]),
           Rule(inport:2, [fwd(1)])]
    register(rules)
def web monitor():
    q = (Select(bytes) *
           Where(inport=2    srcport=80) *
           Every(30))
           q >>    Print()
def main():
    repeater()
    monitor()
```

The composition is very clear. The repeater function only does the repeater's job. The web monitor function is similar. The main function only needs to assemble the components (modules) together. This process helps to isolate the functions from each other effectively. When the programmers amend any of them, they will not affect the other functions. It enables us to swap out the given network monitor easily. And the advantage is that the repeater code or the forwarding logic is no needed to be touched.

6.3.2 Efficient Run-Time System

Every packet is sent to the controller because the flow table of the switch is empty at the very beginning. When receiving a packet, the run-time system traverses the registered forwarding policies and collect a list of actions for that switch. Further, the run-time system has the ability to deal with the rules depends on the real-time traffic. The switch no longer carries complicated functions. The arbitrary functions are implemented with policies on the run-time system which is can support. The rules are customized easily on the run-time system to make the switch more portable and more flexible when facing complicated situations.

According to the newest versions the OpenFlow provides, an interface to the tables in modern will be implemented in the hardware. In future tasks, programmers plan to extend a run-time system to represent sophisticated policies.

6.4 Consistent Updates

The future task is that programs need to transfer frequently from one policy to another and changes in network load due to several reasons. As a high-level network have been designed, the update operations and new flexible policies provide decent guarantees about network behaviors.

6.4.1 Per-Packet Consistent Updates

In fact, a majority of upgrades are possible when the policies are similar. If the new one is comparatively simple or retraction, updates will become simpler and less sophisticated to operate.

6.4.2 Per-Flow Consistency

Although the mechanism like the per-packet consistency is powerful, it is not still defective to deal problems. Sometimes, lots of applications require several streams of related packets. A per-flow update is invented to deal with this. It ensures streams to process with all packets share almost the same configuration. From personal perspective, the reason why implementing is more sophisticated than per-packet consistency is that the belonging to active flows. By letting the old version in place, the run-time system can preinstall the new configuration as in a stable consistency. The controller uses timeouts on the rules for the old configuration and also installs on the lower but new one.

When all of the flows matching one of the same rules complete, the rules for a completely new go into service.

6.5 Summary

To conclude, the Frenetic language provides programmers with several crucial and powerful abstractions. Compiler and run-time system implement these abstractions. Furthermore, exploring ways to raise the level of abstraction is still on progress and have not reached its climax. By this, supporting

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

program modules can make people believe that these abstractions will keep up to break the barriers and create new applications on SDN.

According to the writer of the original paper, the researches about Frenetic are still progressing, but Frenetic is executed to be efficient than any others before.

7. Conclusion

This paper presents the motivation, application, and several evaluations and predictions of Software Defined WAN. The positive effect which SDN brings to us is obvious. In Google's B4 deployment example, SDN helps the companies spend less money on high-end routing gears. The internet utilization is driven to nearly 100%. The separation between software and hardware enables the rapid iteration on application level. Centralized traffic engineering does work as a significant role which allocates constrained network resources among competing services.

However, the growth of SDN also leads to concerns. In "The rise of SDN", the security implementation is needed when an experimental prototype is promoted to reality. In "Promises and Limitations of SDN", the three main features-centralization, flow control, and programmability-performs in real traffic is not as good as designed. With a growing number of SDN deployment, it just changes the commercial competence between companies from the hardware level to the software level. This trend leads to a more complicated network environment.

We should not simply consider SDN as a panacea to the networks. It is just a developing tool which is used to improve the internet environment.

Acknowledgment

The authors wish to thank teaching professor Bill Nace from Carnegie Mellon University.

References

[1] Mckeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008). OpenFlow. ACM SIGCOMM Computer Communication Review, 38(2), 69-74. doi:10.1145/1355734.1355746

[2] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., . . . Vahdat, A. (2013). B4. ACM SIGCOMM Computer Communication Review, 43(4), 3-14. doi:10.1145/2534169.2486019

[3] Russ White and Shawn Zandi. "Cloudy-Eyed: Complexity and Reality with Software-Defined Networks". In: Internet Protocol Journal19.3 (Nov. 2016), pp. 31–41.

[4] Ajay Nehra, Meenakshi Tripathi, and M. S. Gaur. 2017. "Global view" in SDN: existing implementation, vulnerabilities & threats. In Proceedings of the 10th International Conference on Security of Information and Networks (SIN '17). Association for Computing Machinery, New York, NY, USA, 303–306. doi: 10.1145/3136825.3136862

[5] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. 2013. Advanced study of SDN/OpenFlow controllers. In Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '13). Association for Computing Machinery, New York, NY, USA, Article 1, 1–6. doi:10.1145/2556610.2556621

[6] N. Foster et al., "Languages for software-defined networks," in IEEE Communications Magazine, vol. 51, no. 2, pp. 128-134, February 2013, doi: 10.1109/MCOM.2013.6461197.

[7] C. J. Casey, M. Yan, C. Chojnacki and A. Sprintson, "Flowsim: Interactive SDN switch visualization," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015, pp. 34-36, doi: 10.1109/NFV-SDN.2015.7387402.

[8] C. S. Gomes, F. S. Dantas Silva, E. P. Neto, K. B. Costa and J. B. da Silva, "Towards a Modular Interactive Management approach for SDN Infrastructure orchestration," 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, 2016, pp. 1-6, doi: 10.1109/NFV-SDN.2016.7919467.

[9] M. Erel, E. Teoman, Y. Öz çevik, G. Se çinti and B. Canberk, "Scalability analysis and flow admission control in mininet-based SDN environment," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, 2015, pp. 18-19, doi: 10.1109/NFV-SDN.2015.7387396.

[10] C. Tselios, I. Politis and S. Kotsopoulos, "Enhancing SDN security for IoT-related deployments

ISSN 2616-7433 Vol. 3, Issue 1: 113-124, DOI: 10.25236/FSST.2021.030118

through blockchain," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 303-308, doi: 10.1109/NFV-SDN.2017.8169860.
[11] D. Tatang, F. Quinkert, J. Frank, C. Röpke and T. Holz, "SDN-Guard: Protecting SDN controllers against SDN rootkits," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 297-302, doi: 10.1109/NFV-SDN.2017.8169856.
[12] Lili Xiao, Shuangqing Xiang, and Huibiao Zhuy. 2018. Modeling and verifying SDN with multiple controllers. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18). Association for Computing Machinery, New York, NY, USA, 419–422. DOI:https://doi.org/10.1145/3167132.3167381
[13] K. Tantayakul, R. Dhaou and B. Paillassa, "Mobility management with caching policy over SDN architecture," 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 1-7, doi: 10.1109/NFV-SDN.2017.8169830.