International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Software Defined Networking (SDN) (Analysis & Future Direction)

Siva Kumar Mamillapalli

siva.mamill@gmail.com

Abstract

Software-defined networking (SDN) is a modern network architecture that separates the control and management planes from the data plane, aiming to enhance network manageability and programmability. Its programmable nature has led to widespread adoption in academia and industry. However, SDN faces various challenges during implementation and integration with existing technologies. This paper assesses different SDN approaches across seven key areas: network testing, flow rule installation, network security, and controller platforms. Each area significantly influences SDN implementation, ensuring efficient packet handling, secure policies, and effective resource management. Comparative analysis of existing studies offers insights into their classifications, benefits, and limitations, guiding future research directions.

Keywords: SDN, SDN controller platform, Data Plane, Control Plane, Packet Handling, Network Management, Network Testing and Verification, Memory Management, SDN e simulators, SDN programming languages

1. **Introduction**:

Existing network architectures present significant management challenges due to the complex nature of their design. In these traditional networks, the functionalities of the data, control, and management planes are intertwined and closely associated within the network devices themselves. The control plane, utilizing routing protocols, manages the forwarding of data packets according to established network policies. This tight integration of functionalities within the forwarding devices makes network management a complex task, and optimizing network performance becomes a difficult undertaking. Furthermore, the increasing complexity and demands of modern network applications and services necessitate the evolution of the Internet to effectively address these new challenges.

To overcome these limitations and facilitate network evolution, the concept of "programmable networks" has emerged. This concept has been explored through two primary approaches: active networking and programmable networks. Active networking focuses on incorporating network intelligence into the network nodes, enabling them to perform customized operations on packets beyond typical packet processing. Programmable networks, on the other hand, provide the capability to control the behavior of network nodes and manage network flow through software.

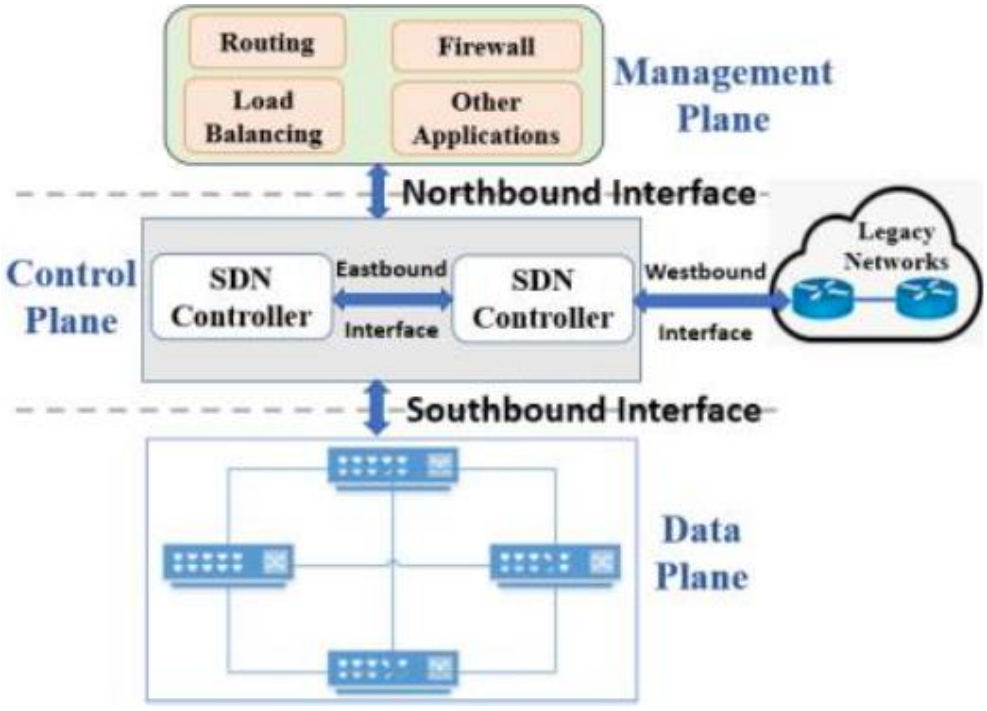
Building upon these ideas, the 4D project introduced a new network design based on four distinct planes: decision, dissemination, discovery, and data. This architecture emphasizes the separation of routing decision-making logic from the protocols that govern communication between forwarding

devices. The decision plane maintains a comprehensive view of the entire network topology and installs configuration commands within the data plane to enable communication. The dissemination and discovery planes provide efficient communication pathways between the decision and data planes. Another significant contribution, Ethane, proposed a novel network architecture specifically designed for enterprises. Ethane allows network managers to configure and control the entire network using a centralized controller.

These research endeavors laid a crucial foundation for the separation of the data and control planes, ultimately leading to the development of software-defined networking (SDN). SDN represents a practical implementation of a set of networking software tools that enable centralized control of a network. While SDN is not the sole solution to embrace separation and programmability, it has gained widespread acceptance in both academic and industry circles due to the rapid advancements occurring in both the control and data planes. To further promote SDN and standardize the OpenFlow Protocol (OFP), a group of network operators, service providers, and vendors established the Open Network Foundation (ONF). In the academic realm, the OpenFlow Network Research Center was established to focus specifically on advancing SDN research.

1.1 Application Programming Interfaces

Application Programming Interfaces (APIs) are essential to Software-Defined Networking (SDN). They facilitate communication between the different planes of the network: data, control, and management. Commonly used APIs include Southbound APIs (SBIs), Northbound APIs (NBIs), and, for distributed controllers, East/Westbound APIs. These APIs are integral architectural elements of SDN, enabling the configuration of both forwarding devices and network applications. A visual representation of the SDN layered architecture, including these APIs.



1.2 Network Configuration and Flow Rules Installation

Traditional computer networks rely heavily on Access Control List (ACL) policies and routing protocols for configuration. ACL policies define the rules that govern how network devices operate, ensuring they meet the requirements of users, applications, and organizations. Routing protocols determine the optimal paths for data transmission between source and destination. In Software-Defined Networking (SDN), ACL policies are configured within the network's control plane. These policies are then used to generate flow rules, which are subsequently installed in the forwarding devices. Because network demands, host requirements, and network topology can change, these policies often require updates to allow or deny specific communications.

SDN programming languages, such as Pyretic, Frenetic, and Maple, provide tools for specifying ACL policies within the application environment. They utilize parallel and sequential composition operators to facilitate the efficient implementation of these policies. When a host initiates communication, the forwarding device (typically a switch) consults its flow table to determine if a relevant flow rule exists. If no matching rule is found, the switch sends a digest packet to the controller. The controller then calculates the optimal path between the source and destination host, taking into account the network topology and the defined ACL policy. Flow rules are installed along this calculated path, and these rules govern the subsequent communication. The switch stores these flow rules in its flow table until a timeout value is reached.

Two types of timeouts are used: soft timeouts and hard timeouts. A soft timeout specifies that a flow rule is removed from the switch's flow table if it remains unused for a predefined period. A hard timeout, on the other hand, dictates that a flow rule is deleted after a specific duration, regardless of activity. These timeout values are dependent on the specific application environment and the controller platform being used.

1.3 SDN Advantages

SDN has many advantages over traditional networks owing to the lower maintenance, ease of management, and implementation of ACL policies. It simplifies network management and control by managing the whole network from the centralized controller. Moreover, forwarding devices (switches) become simplified as network intelligence is shifted to the controller; thus, these devices are left with very basic functionalities as they only need to act according to the instructions from the controller and do not require understanding and processing heterogeneous algorithms and protocols. In addition, the forwarding devices also help the controller for route computations and link/node monitoring, along with other tasks such as network management and diagnostics

2. Network Testing and Verification

Several tools and techniques have been developed to address debugging and testing challenges in both traditional and Software-Defined Networks (SDNs). NDB (Network Debugger) is one such tool designed for debugging SDNs. It operates similarly to the GNU debugger (GDB), allowing developers to set breakpoints, monitor variables with watches, and trace packet backtraces. NDB's architecture

consists of two primary components: a proxy and a collector. The proxy component creates a "postcard" message from information received from the data plane and transmits it to the control plane. The collector, upon receiving these messages, stores the postcards and generates backtraces for the specified data packets. To efficiently manage and retrieve the postcard data, the collector utilizes a hash table data structure.

Veriflow is another tool that focuses on detecting network-wide invariants in real time. It can generate alerts or even block events based on these detections. When the controller generates flow rules, they are forwarded to Veriflow for invariant checking. If a network-wide invariant violation is detected, Veriflow notifies the network administrator or blocks the flow rules. Otherwise, the flow rules are passed on to the data plane. Veriflow's verification process involves three main steps. First, the network is divided into a set of equivalence classes (ECs) based on network routing policies. Second, Veriflow constructs individual graphs for each EC, representing the network behavior within that class. Finally, these graphs are used to determine the status of the network invariant. Veriflow stores network information, such as ACL policies, in trie data structures for efficient access and systematically computes the ECs.

Beyond NDB and Veriflow, other research efforts have also contributed to addressing debugging and testing issues in both traditional and SDN environments. These works have explored areas such as detecting network anomalies, ensuring data plane consistency, and resolving conflicts between different network applications to enable parallel execution.

3. Flow Rule Installation Mechanisms

Several frameworks and mechanisms have been developed to optimize flow rule placement and management in networks. ORPP (Optimal Rule Placement Protocol) offers two such frameworks: OFFICER and aOFFICER. These frameworks assist in defining and installing flow rules within the data plane, adhering to both technical and non-technical requirements. OFFICER is designed for defining and installing flow rules for a known set of requirements within a specific time interval. aOFFICER, on the other hand, handles the computation and installation of flow rules for unknown and dynamically changing requirements over a given time interval. Both frameworks offer effective solutions for flow rule placement.

vCRIB provides a mechanism that offers an abstraction layer for specifying and managing flow rules, particularly for network operators in data center environments. It also addresses performance and scalability concerns by facilitating the partitioning and installation of flow rules at both hypervisors and switches.

DevoFlow introduces a modified OpenFlow model that allows network operators to concentrate on essential flow rules for network management. It achieves this by decoupling network control from global visibility. This decoupling reduces internal communication between the control and data planes in two primary ways. First, it minimizes the transfer of statistics for less critical flows. Second, it reduces the need to involve the control plane for most flow setups. By minimizing communication overhead, DevoFlow maintains a reasonable level of visibility. However, it's important to note that the DevoFlow prototype has not yet been tested with actual network traffic.

4. SDN Controller Platforms

Software-Defined Networking (SDN) controllers are the central intelligence of the network, acting as a strategic control point. They comprise a collection of modules capable of performing various network tasks, such as managing network topology and gathering network statistics. Network applications, including network policies, are installed on these controllers to govern data communication between end nodes. This section examines several controller platforms.

Beacon is a Java-based controller employing a centralized architecture. It uses custom-designed northbound and southbound APIs, supporting OpenFlow 1.0, and offers both command-line interface (CLI) and web user interface (WebUI) access. Beacon also features multi-command-line threading and modularity. It serves as the foundation for Floodlight and prioritizes developer-friendliness, high performance, and the ability to start and stop existing processes. Beacon has contributed to the exploration of OpenFlow controller design.

Beehive is a distributed control platform written in the Go programming language, featuring a distributed hierarchical architecture. It utilizes REST APIs for both northbound and southbound communication, supporting OpenFlow 1.0 and 1.2. Beehive runs on the Linux platform and provides CLI access.

DCFabric is implemented using C and JavaScript and has a centralized architecture. It uses REST APIs for northbound and southbound communication, supporting OpenFlow 1.3. DCFabric operates on Linux and supports both CLI and WebUI. It also features multi-threading, good modularity, and strong consistency.

Disco is a Java-based controller with a distributed flat architecture. It uses REST APIs for northbound, southbound, and east/westbound communication, supporting OpenFlow 1.0 and AMQP, respectively. Disco uses proprietary licenses, has good modularity, but limited documentation.

Faucet is implemented in Python and has a centralized architecture. It utilizes a southbound API with OpenFlow 1.3. Running on the Linux platform, Faucet offers both CLI and WebUI access. It uses the Apache 2.0 license, supports multi-threading, and provides good consistency.

5. Conclusion

Software-Defined Networking (SDN) has revolutionized network management by centralizing control in a single controller. This architectural shift offers numerous advantages, including increased programmability, accelerated innovation, and simplified security policy implementation. This paper provides a concise overview of traditional networking and SDN, covering their background, Application Programming Interfaces (APIs), network configurations, and the advantages offered by the SDN paradigm. The paper is structured into seven key areas: network testing and verification, flow rule installation mechanisms, network security and management challenges in SDN, memory management studies, SDN simulators and emulators, SDN programming languages, and SDN controller platforms. Each of these categories is explored in detail, including their implementation mechanisms. We analyze these mechanisms by summarizing and comparing the various techniques, highlighting the lessons

learned from each. Furthermore, we analyze and discuss recent research, comparing it with the work presented in this paper. Finally, we offer comprehensive guidelines for future research directions and conclude the paper.

6. References

1.Benson T., Akella A., Maltz D. Unraveling the complexity of network management; Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI'09; Berkeley, CA, USA. 22–24 April 2009; pp. 335–348. [Google Scholar]
2.Tennenhouse D.L., Smith J.M., Sincoskie W.D., Wetherall D.J., Minden G.J. A survey of active network research. IEEE Commun. Mag. 1997;35:80–86. doi: 10.1109/35.568214. [DOI] [Google Scholar]
3.Campbell A.T., Meer H.G.D., Kounavis M.E., Miki K., Vicente J.B., Villela D. A survey of programmable networks. SIGCOMM Comput. Commun. Rev. 1999;29:7–23. doi: 10.1145/505733.505735. [DOI] [Google Scholar]
4.Rexford J., Greenberg A., Hjalmtysson G., Maltz D.A., Myers A., Xie G., Zhan J., Zhang H. Network-wide decision making: Toward a wafer-thin control plane; Proceedings of the Third Workshop on Hot Topics in Networks HotNets-III; Chicago, IL, USA. 17–22 August 2014; pp. 59–64. [Google Scholar]
5.Greenberg A., Hjalmtysson G., Maltz D.A., Myers A., Rexford J., Xie G., Yan H., Zhan J., Zhang H. A clean slate 4D approach to network control and management. ACM SIGCOMM Comput. Commun. Rev. 2005;35:41–54. doi: 10.1145/1096536.1096541. [DOI] [Google Scholar]
6.Caesar M., Caldwell D., Feamster N., Rexford J., Shaikh A., van der Merwe J. Design and implementation of a routing control platform; Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation; Berkeley, CA, USA. 2–4 May 2005; pp. 15–28. [Google Scholar]
7.Casado M., Freedman M.J., Pettit J., Luo J., McKeown N., Shenker S. Ethane: Taking control of the enterprise. ACM SIGCOMM Comput. Commun. Rev. 2007;37:1–12. doi: 10.1145/1282427.1282382. [DOI] [Google Scholar]
8.Open Networking Foundation . Available online: https://www.opennetworking.org/about.
9.Nick M., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. 2008;38:69–74. [Google Scholar]
10.Open Networking Research Center (ONRC) . Available online: http://onrc.net.
11.Feamster N., Rexford J., Zegura E. The road to SDN: An intellectual history of programmable networks. ACM SIGCOMM Comput. Commun. Rev. 2014;44:87–98. doi: 10.1145/2602204.2602219. [DOI] [Google Scholar]
12.Mckeown N. How SDN Will Shape Networking.. Available online: http://www.youtube.com/watch?v=c9-K5OqYgA.
13.Alvigzu R., Maier G., Kukreja N., Pattavina A., Morro R., Capello A., Cavazzoni C. Comprehensive survey on SDN: Software defined networking for transport networks. IEEE Commun. Surv. Tutor. 2017;19:2232–2283. doi: 10.1109/COMST.2017.2715220. [DOI] [Google Scholar]

14.Mousa M., Bahaa-Eldin A.M., Sobh M. Software defined networking concepts and challenges; Proceedings of the 11th International Conference on Computer Engineering & Systems (ICCES); Cairo, Egypt. 20–21 December 2016; pp. 79–90. [Google Scholar]
15.Schenker S. The Future of Networking, and the Past of Protocols. 2011. Available online: http://www.youtube.com/watch?v=YHeyuD89n1Y.
16.Zohaib L., Sharif K., Li F., Karim M.M., Biswas S., Wang Y. A comprehensive survey of interface protocols for software defined networks. J. Netw. Comput. Appl. 2020;156:102563. [Google Scholar]
17.Curtis R.A., Mogul J.C., Tourrilhes J., Yalagandula P., Sharma P., Banerjee S. DevoFlow: Scaling flow management for high-performance networks. ACM SIGCOMM Comput. Commun. Rev. 2011;41:254–265. doi: 10.1145/2043164.2018466. [DOI] [Google Scholar]