BlockFlow: A decentralized SDN controller using blockchain

Theviyanthan Krishnamohan[1], Kugathasan Janarthanan[1], Peramune P.R.L.C[1], Ranaweera A.T[1]
Sri Lanka Institute of Information Technology (SLIIT), Department of ISE – Sri Lanka, Malabe

Email: { it14004414, it15051608, it14009532, it15026644}@my.sliit.lk

DOI: 10.29322/IJSRP.10.03.2020.p9991
http://dx.doi.org/10.29322/IJSRP.10.03.2020.p9991

***Abstract*—With the rise of cloud computing, data centers, and big data, the current rigid network architecture has been found to be inadequate. The modern technological demands require a flexible and easily reconfigurable network architecture. Software Defined Networking is a revolutionary concept that separates the control plane of network devices from their data plane and centralizes the control plane of all devices, facilitating the controlling of the entire network through a single portal. This helps us create flexible network architectures that can be reconfigured quickly to fit different needs. However, centralizing control leads to a Single Point of Failure and makes the network vulnerable to Denial of Service attacks, which is one of the major reasons why industries are reluctant to adopt this technology. Blockchain provides us a with a distributed ledger and a decentralized state, allowing us to create decentralized applications that run over multiple computers. This research aims to distribute the control plane of Software Defined Networks across multiple devices using blockchain. This addresses the existing security vulnerabilities of the Software Defined Network architecture such as Single Point of Failure while continuing to keep the control plane logically centralized, thereby allowing the network to be configured through a single portal. The resulting architecture has a physically distributed control plane whose logic is centralized.**

Keywords—blockchain, software defined networking, decentralized applications, sdn controllers

I. INTRODUCTION

Traditional network devices have a tightly coupled data plane and control plane. Therefore, rules on how to handle packets and where to forward them, and the algorithm that produces such rules are all found in one device.

This means that to reconfigure a network, one must reconfigure each device in the network. Since under various circumstances networks need to be able to be reconfigurable without being highly labor intensive, such an architecture has proven to be ineffective and inefficient [1].

Owing to modern demands such as cloud computing and the need to be independent of network device vendors to implement the protocols and capabilities that may be needed by different companies [2], an easily configurable and controllable network paradigm was needed.

Software Defined Networking (SDN) is a concept that was conceived to solve the problem of excessive dependence on network device vendors, and the inability to make network-wide configurational changes easily.

SDN decoupled the data plane and control plane of network devices, centralized the control plane and placed it out of network devices allowing network administrators to not only make network-wide configurational changes through a central point but also implement the desired routing algorithms, protocols and policies irrespective of whether network device vendors support them.

This has allowed networks to be flexible and easily reconfigurable helping them change quickly to cater to different use cases and ensure Quality of Service [3]. However, the centralization of the control plane introduced new challenges such as Single Point of Failure and vulnerability to Denial of Service attacks [4]. In addition, SDN also creates security issues such as lack of authentication between applications and the controller, absence of encryption between controller and switches, and weak authentication between the controller and the switches [5].

Blockchain helps us create trustless consensus among different nodes, and a distributed ledger allowing us to build decentralized applications that run across a network of computers [6]. Using Blockchain to build a decentralized controller application for SDN solves the following issues.

1. Single point of failure as the controller application will be running over multiple computers
2. Denial of Service attack since hijacking just one controller will not suffice for disrupting the network.
3. Application-controller authentication as blockchain uses private-public key pairs

Thus, an SDN controller that is physically distributed but logically centralized will provide the same benefits of a typical SDN architecture while addressing the major security issues of SDN.

The scope of this research is limited to building a controller application logic that would perform basic network functions such as routing.

This paper is composed of the following sections: Section 2 briefs about the related works done, section 3 gives a primer on SDN, section 4 discusses blockchains and their relevance

to this research, section 5 explains how blockchain is incorporated in to the SDN architecture, section 6 describes the methodology, section 7 deals with the tests and results, section 8 briefs about future works, and section 8 provides a conclusion.

II. Related work

Researches in the past have tried to create a multi-controller SDN architecture. Examples of such works are ONIX [7], HyperFlow [8], ONOS [9], DISCO [10], ELASTICON [11], KANDOO [12], and ORION [13] . But such architectures suffer from many problems such as developing an efficient communication between multiple controllers, creating an adequate network design, and integrating new applications that work with the northbound APIs of multiple controllers [14]. Little work has been done to explore the possibility of utilizing the distributed state management capabilities afforded by blockchain to create a multi-controller SDN architecture. Even though C. Tesilos et al. studied the possibility of using blockchain to authenticate and store date from IoT devices using SDN, no architectural changes to SDN to mitigate problems arising from centralized control was proposed [14]. ChainGuard uses SDN to create a firewall for a blockchain network [15]. VeidBlock tries to verify IoT sources using SDN and blockchain [16]. DistBlocknet is the only product that tries to utilize blockchain to distribute an SDN controller, but it is done specifically for IoT networks with the intention of enabling trustless communication between IoT devices.

III. Software defined networks

SDN consists of three layers, viz., the application layer, the control layer, and the infrastructure layer [17]. The infrastructure layer consists of the data-forwarding devices that forward packets based on the flow rules.

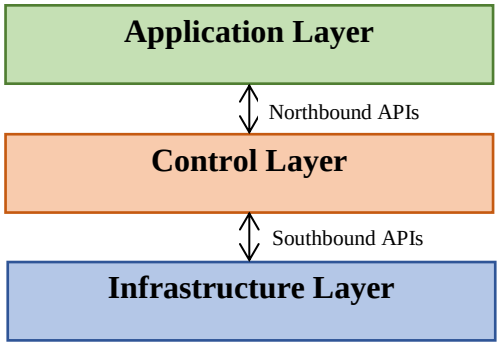

Figure 1 SDN architecture

The control layer consists of the controller device that acts as the interface between the application layer and the infrastructure layer. It is through this layer that network applications that want to control the network can communicate with the infrastructure layer. The controller is typically a server that runs an SDN controller software [18].

The application layer consists of networking applications that program the network infrastructure [19]. This can include routing applications that build routes based on which the data-forwarding devices forward data, firewall applications that control access to different parts of the network, and security monitoring applications.

The control layer provides two Application Programming Interfaces, namely, the northbound APIs and the southbound APIs. The northbound APIs allow the application layer to communicate with the control layer, whereas the southbound APIs provides a communication channel between the control layer and the infrastructure layer.

IV. Blockchain

Blockchain is a combination of different algorithms that uses a mixture of hashing and asymmetric encryption to create a distributed ledger, and trustless communication between different nodes in a network. This technology was introduced through Bitcoin which proposed a peer-to-peer transaction system without a trusted intermediary [20].

In blockchain, transactions between individuals are verified using a private-public key pair, and hashes. A previous transaction is hashed and the hash along with the public key of the receiver of the new transaction is encrypted by the sender's private key to create the signature of the sender. Thus, by verifying the signed transaction using the public key found in the previous transaction (that of the sender who was the receiver in the previous transaction), a transaction can be verified.

A bunch of these transactions is hashed together to form a Merkle tree, and the root hash of the tree is stored in a timestamped block. This block includes the hash of the previous block and the succeeding block will contain the hash of the current block. This way a chain of blocks is formed. If one of the blocks is mutated, then the hash of it will change. Since the hash of this block is a part of the hash of the succeeding block, and since every succeeding block includes the hash of the previous block, mutating one block will make the whole chain that follows the mutated block invalid.

A proof of work algorithm is used to ensure that creating a block results in a cost to the creator of the block. Thus, mutating multiple blocks in a blockchain can be made costly to the point that the cost of fraudulently modifying a blockchain is much more than the supposed benefits. This helps us create an immutable blockchain.

Creators of blocks, called miners, are required to produce a hash that is less than or equal to a certain target. Miners try to achieve this by adjusting a value called nonce in the block. Since it is computationally difficult to reverse engineer a hash, miners are forced to use the trial-and-error method to try to produce a valid hash. This is an expensive process as far as CPU power is concerned. Miners are rewarded for expending their CPU power with bitcoins.

This network of miners helps create a decentralized system that allows trustless communication between individuals while allowing to store data in a distributed ledger, i.e., the blockchain.

Though meant to create a transaction system initially, blockchain can also be used to create decentralized applications that run across multiple nodes.

V. Blockflow architecture

The SDN architecture is slightly modified by using a blockchain network in the control layer to create a decentralized SDN architecture. A decentralized SDN

controller application is built and deployed among several blockchain nodes. Thus, multiple nodes in the network work together to create a combined effect of a controller.

The global state of the network is stored in a ledger distributed among the nodes. Blockchain ensures that the states stay synchronized across multiple nodes. The blockchain also offers JSON based Remote Procedure Calls that allow applications to interact with the blockchain. This is used to create the northbound and southbound APIs.

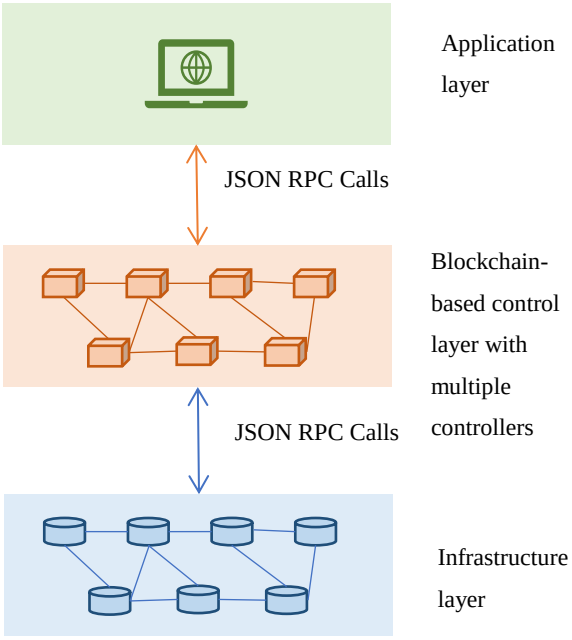

Figure 2 The architecture of BlockFlow

A network application can compute the routes and create a global flow table and write it to the blockchain, which will then be distributed among all the nodes. Data-forwarding devices can then read their respective flows from the blockchain and update their own flow table accordingly.

VI. METHODOLOGY

Ethereum is used to implement the blockchain network. A smart contract is created using Solidity containing the controller application logic. This smart contract is then deployed on a private Ethereum network.

A. *Smart contract*

The smart contract acts both as a database and application logic. Mainly two sets of data are stored in the database:

1. Switches
2. Routes

The switches dataset stores the following data:

1. Switch ID
2. Switch Name
3. Version

The routes dataset is essentially a flow table with the following fields:

1. Route ID
2. Switch ID
3. Next Hop IP address
4. Destination IP address

Each data-forwarding device, called a switch, will have an entry in the switches dataset. The Switch ID is used to uniquely identify each switch in the network. The version of the switch is used to find whether routing for the concerned switch has undergone changes. The default version is set to one and the number is incremented every time a routing entry pertaining to that switch is added, deleted or updated.

The route ID in the routes dataset is used to uniquely identify each routing entries. The switch ID specifies the switch to which the routing entry belongs. The destination IP address stores the IP address to which a packet is meant for. The next hop IP address stipulates the IP address of the port to which a packet should be forwarded. Even though the routes dataset is limited to just 4 datasets owing to the limits of the scope of this research, it can be further expanded to allow switches to function as firewalls.

The smart contract allows the following switch-related functions:

- Create a switch
- Update a switch
- Read a switch
- Delete a switch

The following routes-related functions are allowed:

- Create a route
- Read a route
- Update a route
- Delete a route

B. *Switch Controller*

A Python-based application is built to control a traditional network router in the infrastructure layer. However, this application can be bundled together with any switch software such as OpenVSwitch to work with the proposed architecture.

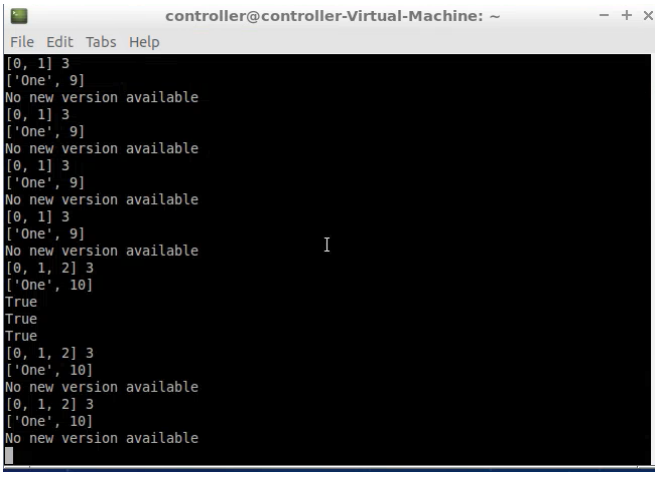

Figure 3 The Command Line Interface of the Switch Controller

This Python application uses Web3.py library and the Application Binary Interface provided by it to interact with a smart contract. An ID can be assigned to each switch using this application. The application, then, will query the smart contract to see if a new version of routing entries for the concerned switch is available. If available, the application will then update the routing table of the network device.

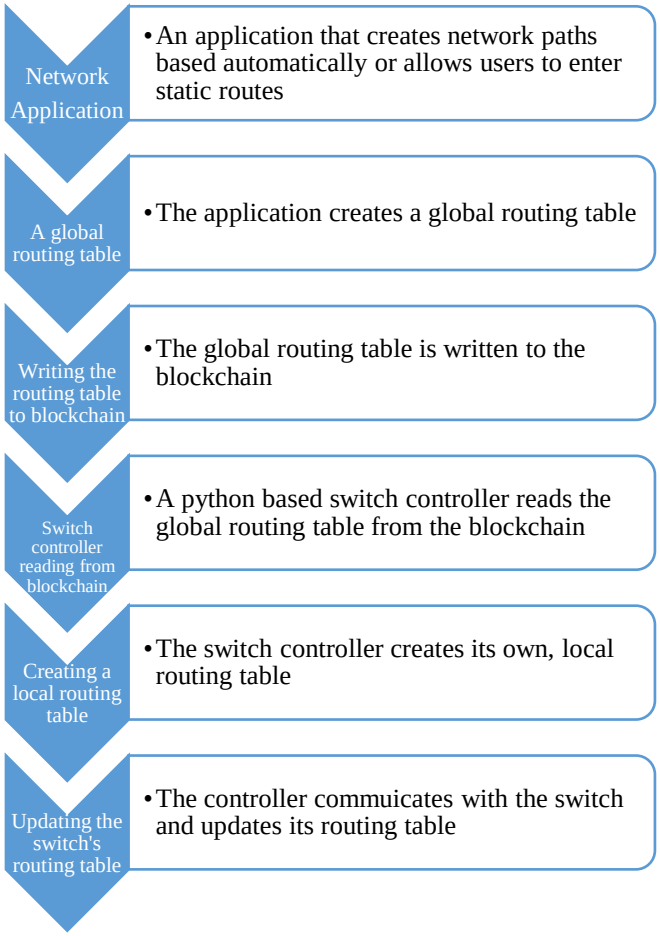

Figure 4 The process workflow of BlockFlow

C. *Ethereum Virtual Machine*

A valid user or application can add or modify the switches and routes dataset in the smart contract by performing a signed transaction. The ether required to perform the transaction can be gained by mining blocks in the Ethereum network using the same account.

The smart contract is deployed in an Ethereum Virtual Machine. This virtual machine is the logical collection of all nodes in the blockchain network.

The Ethereum blockchain network is run using the geth command line interface written using Go. Once the network is created any number of nodes can be added to the network as the administrators see fit.

The communication between the switches and the controllers is outbound, i.e., the network among them is independent of the SDN network.

D. *Application Layer*

The smart contract provides an Application Binary Interface that allows applications written languages like Python, Java, JavaScript, and C# to interact with the smart contract. The applications can make JSON Remote Procedure calls to read or mutate the state of the Ethereum virtual machine.

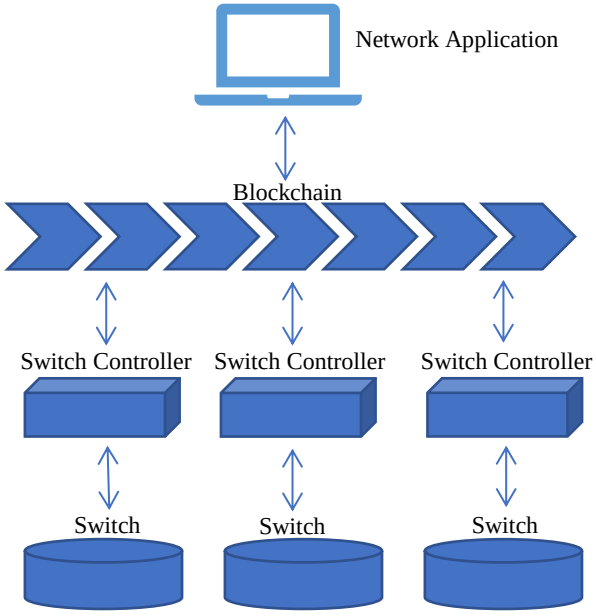

Figure 5 A typical network setup of BlockFlow

VII. Test Setup and Results

A test network was created virtually on a Windows 10 Pro PC (build 1809) with an Intel Core i7-8550U 1.8GHz CPU and 16 GB of RAM. Microsoft's Hyper-V was used to create a virtual network. The Ethereum nodes were run on Lubuntu 18.10 with 256 MB of RAM and one vCPU.

VyOS, an Open Source Linux-based Network OS, was used to create network switches. The Python-based switch controller controlled these switches via SSH.

A React.js-based Single Page Application was created to function in the application layer. This was a simple application that allowed the creation, deletion, and modification of routes and switches.

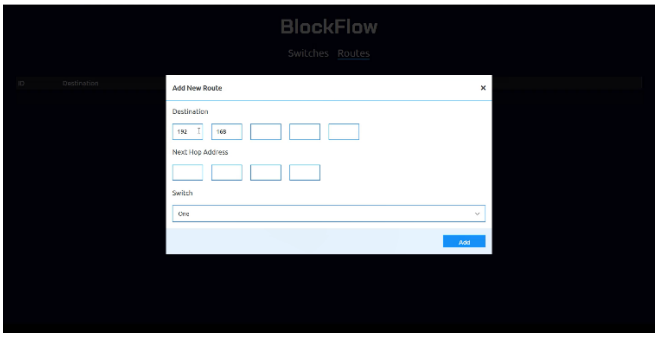

Figure 6 The GUI of the Single Page Application

The following network was created to test the SDN architecture.

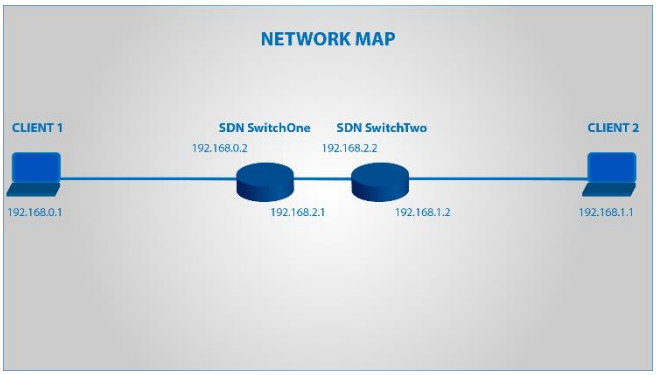

Figure 7 Test network

The network switches were connected together and a Linux client was attached to each of those two switches. ICMP packets, using ping, were sent from one client to another to test connectivity. As the routes dataset was empty in the smart contract, pings kept failing.

Then, routing entries to route packets from one network to the other on each switch were created and the network convergence time was measured. This was repeated 10 times to find the average time. The average convergence time was found to be 14.3 seconds.

The measured time is in consistent with Ethereum's block time of 10 to 19 seconds [21]. However, in order to discourage users from using the proof work algorithm, Ethereum introduced a function called the difficulty bomb. This would result in the block time increasing exponentially every 100,000 blocks.

The difficulty of compromising the SDN network can be calculated as follows.

The hash rate of a controller is the number of hashes a controller can produce per second. The total hash rate of a blockchain network is given by the equation:

$$\mathrm{H} = \sum_{i=1}^{n} h_i$$

Where:

H is the total hash rate of the controllers in the network

n is the total number of controllers in the network

h_i is the hash rate of a controller

According to Bitcoin's whitepaper, the probability P of an attacker compromising the network, when the probability of an authentic controller mining the next block is A, and the probability of a rogue controller mining the next block is Q [20]:

$$P = 1 \; if \; Q \geq A$$

The probability Q of a rogue controller mining the next block is given by

$$Q = \frac{h_q}{H}$$

The probability A of an authentic controller mining the next block is given by

$$A = \frac{h_a}{H}$$

Where:

h_q is the total hash rate of the rogue controllers

h_a is the total hash rate of the authentic controllers

H is the total hash rate of both the authentic and rogue controllers.

Therefore, if Q is to be greater than or equal to A, then the total hash rate of the rogue controllers (h_q) should be more than 50% of the total hash rate (H) of the controller in the network. Thus, the threshold of compromisation C of a network can be defined in terms of H as:

$$C = \frac{1}{2}H$$

So, an attacker needs to have a total hash rate exceeding C to be able to successfully compromise a network.

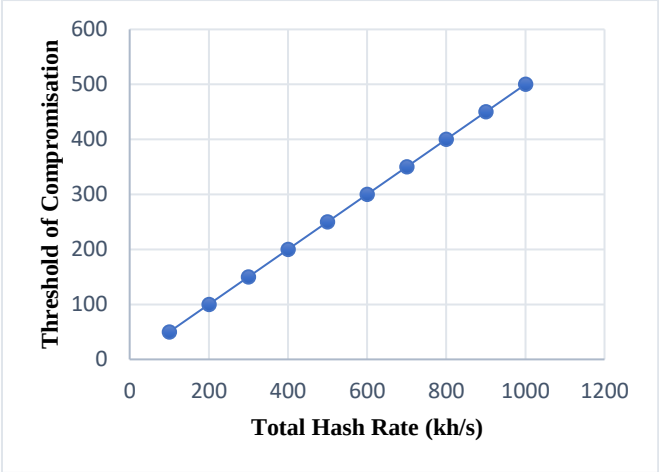

Figure 8 Line graph showing the linear relationship between the total hash rate and the threshold of compromisation (C)

There is a linear relationship between the threshold of compromisation and the total hash rate of the network. Thus, the higher the hash rate, the more difficult it is to compromise a network.

The hash rate of a network can be increased both vertically—by adding a more powerful CPU to a controller, and horizontally—by increasing the number of controllers. Increasing the hash rate vertically is less secure since an attacker can gain access to a high hash rate by simply compromising one controller.

Using blockchain to create a distributed database afforded additional advantages such as synchronization of states across multiple controllers, distributing the workload of the Ethereum virtual machine across all devices, and creating an immutable table of flow entries. Storing state information on the blockchain also ensured the history of changes was available for security auditing and analysis.

VIII. FUTURE WORK

This research could be expanded further by adding a topology discovery logic to the smart contract. At present, devices are added manually through the React based application. However, network switches, on connecting to the control layer, can be allowed to add themselves along with the details of the neighboring switches to a list of connected switches in the smart contract. Then, the application layer may approve the switch and add it to the topology.

In the longer run, the difficulty bomb function as discussed in the above section can make mining blocks increasingly consume more time and may lead to a phenomenon called the ice age [22]. Even though this will no more be a problem once Ethereum adopts the proof of stake algorithm, a custom Ethereum client can be built by modifying the difficulty function to avoid this problem.

IX. Conclusion

This research aimed to solve the problems of Single Point of Failure, Denial of Service attacks, and absence of authentication between the application and the controller. By distributing the control plane of SDN across multiple devices while keeping it logically centralized allowed us to solve the aforementioned problems. In addition, using blockchain also helped resolve the usual challenges posed when trying to use a multi-controller setup such as:

1. State synchronization across devices
2. Even distribution of workload over all controllers
3. An immutable database of flow entries
4. A history of state changes for security auditing and analysis

By offering a solution to the security problems found in SDN using blockchain, this research helps alleviate the security-related skepticism against SDN and facilitates the industry adoption of this technology by network engineers.

References

[1] Q. H. K. B. Fei Hu, "A Survey on Software-Defined Network (SDN) and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 2181-2206, 2014.

[2] D. C. a. J. R. Amin Vahdat, Interviewee, *A Purpose-built Global Network: Google's Move to SDN.* [Interview]. 11 December 2015.

[3] Cisco, "Software-Defined Networking: Why We Like It and How We Are Building On It".

[4] E. A. A. G. M. K. K. M. I. a. S. G. Adnan Akhunzada, "Securing Software Defined Networks: Taxonomy, Requirements, and Open Issues," *IEEE Communications Magazine* , vol. 53, no. 4, pp. 36-44, 2015.

[5] L. Gupta, "SDN: Development, Adoption and Research Trends," 20 December 2013. [Online]. Available: https://www.cse.wustl.edu/~jain/cse570-13/ftp/sdn.pdf. [Accessed 30 December 2018].

[6] M. Swan, Blockchain: Blueprint for a New Economy, Sebastopol: O'Reilly, 2015.

[7] M. C. N. G. J. S. L. P. M. Z. R. R. Y. I. H. I. T. H. S. S. Teemu Koponen, "Onix: A Distributed Control Platform for Large-scale Production Networks," *OSDI* , vol. 10, pp. 1-6, 2010.

[8] A. a. Y. G. Tootoonchian, "Hyperflow: A distributed control plane for openflow," *Internet network management conference on Research on enterprise networking,* pp. 3-3, 2010.

[9] P. a. G. M. a. H. J. a. H. Y. a. K. M. a. K. T. a. L. B. a. O. B. a. R. P. a. S. W. a. o. Berde, "ONOS: towards an open, distributed SDN OS," *Proceedings of the third workshop on Hot topics in software defined networking,* pp. 1-6, 2014.

[10] K. a. B. M. a. L. J. Phemius, "Disco: Distributed multi-domain sdn controllers," in *Network Operations and Management Symposium (NOMS),* 2014.

[11] A. a. H. F. a. M. S. a. L. T. a. K. R. R. Dixit, "ElastiCon- an elastic distributed SDN controller," *Architectures for Networking and Communications Systems (ANCS),* pp. 17-27, 2014.

[12] S. a. G. Y. Hassas Yeganeh, "Kandoo: a framework for efficient and scalable offloading of control applications," *Proceedings of the first workshop on Hot topics in software defined networks,* pp. 19-24, 2012.

[13] J. B. K. G. Z. C. J. W. a. B. H. Y. Fu, "Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks," *IEEE 22nd International Conference on Network Protocols (ICNP),* pp. 569-576, 2014.

[14] M. B. M. a. R. B. Othmane Blial, "An Overview on SDN Architectures with Multiple Controllers," *Journal of Computer Networks and Communications,* vol. 2016, 2016.

[15] I. P. a. S. K. C. Tselios, "Enhancing SDN Security for IoT-related deployments through Blockchain," *Third International Workshop on Security in NFV-SDN,* pp. 303-308, 2017.

[16] S. H. a. R. S. Mathis Steichen, "ChainGuard - A Firewall for Blockchain Applications using SDN with OpenFlow," *2017 Principles, Systems and Applications of IP Telecommunications (IPTComm),* pp. 1-8, 2017.

[17] A. G. a. K. Z. Abbasi, "VeidBlock: Verifiable identity using blockchain and ledger in a software defined network," *SCCTSA2017 co-located 10th IEEE/ACM Utility and Cloud Computing Conference,* pp. 173-179, 2017.

[18] S. D. O. K. E.H. Haleplidis, "SDN Layers and Architecture Terminology," 15 July 2013. [Online]. Available: https://tools.ietf.org/id/draft-haleplidis-sdnrg-layer-terminology-00.html. [Accessed 1 January 2019].

[19] N. &. A. T. &. B. H. &. P. G. &. L. P. L. &. R. J. &. S. S. &. T. J. McKeown, "OpenFlow: Enabling Innovation in Campus Networks," *Computer Communication Review,* vol. 38, pp. 69-74, 2008.

[20] S. S. a. S. S.-H. a. P. K. C. a. B. F. a. D. L. a. J. F. a. N. V. a. M. M. a. N. Rao, "Are We Ready for SDN? Implementation Challenges for Software-Defined Networks," *IEEE Communications Magazine,* vol. 51, pp. 36-43, 2013.

[21] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed 1 January 2019].

[22] [Online]. Available: https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/consensus.go. [Accessed 2 January 2019].

[23] D. G. WOOD, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER," 2018. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf. [Accessed 3 January 2019].