






## Article

# Performance Evaluation of Stateful Firewall-Enabled SDN with Flow-Based Scheduling for Distributed Controllers

Senthil P. <sup>1</sup>, Balasubramanian Prabhu Kavin <sup>2</sup>, S. R. Srividhya <sup>3</sup>, Ramachandran V. <sup>4</sup>, Kavitha C. <sup>3,\*</sup>  
and Wen-Cheng Lai <sup>5,6,\*</sup>

- <sup>1</sup> Department of Computer Science and Engineering, Karpagam College of Engineering, Myleripalayam Village, Othakkal Mandapam 641032, Tamil Nadu, India
  - <sup>2</sup> Department of Data Science and Business Systems, College of Engineering and Technology, SRM Institute of Science and Technology, SRM Nagar, Chengalpattu District, Kattankulathur 603203, Tamil Nadu, India
  - <sup>3</sup> Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai 600119, Tamil Nadu, India
  - <sup>4</sup> Department of Computer Science and Engineering, Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad 500090, Telangana, India
  - <sup>5</sup> Bachelor Program in Industrial Projects, National Yunlin University of Science and Technology, Douliu 640301, Taiwan
  - <sup>6</sup> Department Electronic Engineering, National Yunlin University of Science and Technology, Douliu 640301, Taiwan
- \* Correspondence: kavitha4cse@gmail.com (K.C.); wenlai@yuntech.edu.tw (W.-C.L.)

**Abstract:** Software-defined networking (SDN) is a network approach achieved by decoupling of the control and data planes. The control plane is logically centralized and the data plane is distributed across the network elements. The real-time network is in need of the incorporation of distributed controllers to maintain distributed state information of the traffic flows. Software-based solutions aid distributed SDN controllers to handle fluctuating network traffic and the controller's configurations are dynamically programmed in real time. In this study, SDN controllers were programmed with a stateful firewall application to provide firewall functionalities without the support of committed hardware. A stateful firewall filtered traffic based on the complete context of incoming packets; it continuously evaluated the entire context of traffic flows, looking for network entry rather than specific traffic flows. In addition, a flow-based scheduling module was implemented in the distributed controllers to improve network scalability. A network cluster was configured with three distributed controllers and we experimented with three independent network topologies. The performance of the proposed network model was evaluated by measuring and analyzing metrics such as network throughput (kbps), delay (ms) and network overhead (pkt/ms) for various combinations of controllers and topologies. The results of the analysis were determined using the mininet emulator. The findings of the performance evaluation indicate that the distributed SDN controllers performs better than a centralized controller. When comparing distributed SDN with two controllers and distributed SDN with three controllers the overall network throughput is increased by 64%, the delay is decreased by 43% and network overhead is reduced by 39%.

**Keywords:** software defined network; stateful firewall; SDN; distributed controller; SDN performance analysis and flow-based scheduling



**Citation:** P., S.; Kavin, B.P.; Srividhya, S.R.; V., R.; C., K.; Lai, W.-C. Performance Evaluation of Stateful Firewall-Enabled SDN with Flow-Based Scheduling for Distributed Controllers. *Electronics* **2022**, *11*, 3000. <https://doi.org/10.3390/electronics11193000>

Academic Editors: Celestine Iwendi and Thippa Reddy Gadekallu

Received: 30 August 2022

Accepted: 19 September 2022

Published: 22 September 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

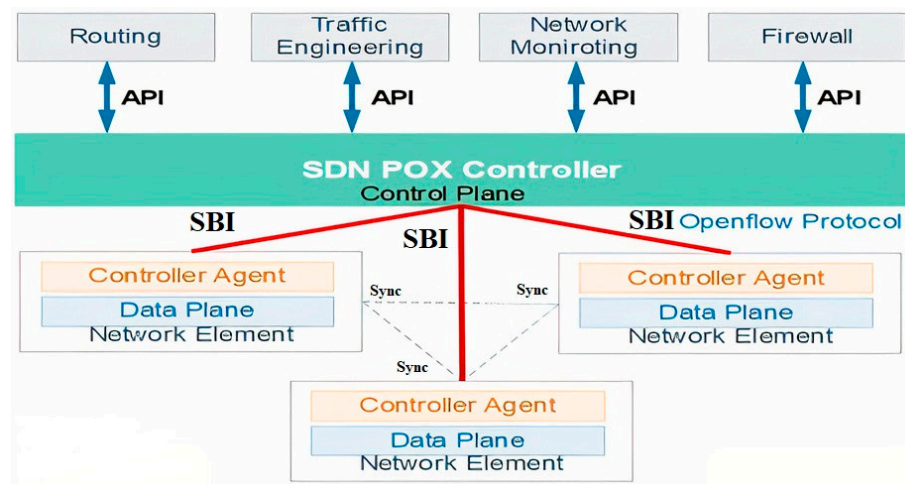
Nowadays, a network of devices influences our way of doing business and our way of living. Computer networks play a major role in everyday activities allowing users and their devices to connect. Recently, there has been an explosion of business applications allowing for dynamic collaboration among clients. Each of these businesses is different, but all rely on some form of a computer network to exchange information and data. More and

more organizations are deploying information technology and working as part of a virtual team, giving their employees flexibility. This may mean they are performing complex work without having to join regular meetings and brainstorming sessions. Virtual teams tend to create independent groups that operate independently of each other, which creates real freedom for workers. However, as easy as it may be to work remotely for a business, it is equally important to create policies that ensure the safety of the whole team. These policies need to be established well in advance and adhere to all standards and guidelines.

By linking devices, data, and computers, interactions across remote users can be accomplished through a method known as networking. This makes it easier for users to share resources and information. To enterprises and individuals, networks are a crucial factor as they dramatically increase business activities. As internet-connected devices are growing daily with more and more information being transmitted between devices, it becomes harder to scale up the current network. Network growth is taking place to satisfy the demand, with almost all of the efforts going into manually configuring each switch and router. Many network features are configured throughout network devices in traditional networking, such as switches, routers, firewalls, and controllers. It becomes difficult to configure and reconfigure the network devices based on the fixed-set rules to adapt to varying traffic, errors and other network adjustments. Commonly, vendors implement and manage control planes in conventional networking systems. This makes the network equipment dependent on a specific vendor and forces the service provider to manually configure every piece of network equipment. In addition, network operators may fail to maintain the high reliability of the network.

Software-defined networking (SDN) is a distinctive network structure that centrally manages all network equipment to enable adaptive service control, faster time to market, and full automation of the network stack. The key to an efficient SDN architecture is a flexible, fault-tolerant infrastructure that can be dynamically reconfigured to meet changing network demands. Ideally, every virtual switch must have its own local SDN network to provide a high-level service that is defined at the network device layer and that can be configured by software. Network systems have traditionally been designed around a dedicated centralized network operating system and clustered control computers that handle the networking and switching functions. When SDN-enabled switches are used together with a distributed architecture, centralized management becomes difficult or impossible. To provide end users with choice, network operators need an integrated, software-based SDN architecture that can provide this centralized management capability with a high degree of flexibility. To address this challenge, industry leaders are developing new SDN systems with a fully open architecture. SDN systems that employ cloud-based SDN orchestration, or that adopt OpenFlow to perform network functions such as switching, routing, and security, provide the benefits of a centralized management platform. This level of flexibility is not possible with network operating systems. The service assurance component within SDN allows the network to self-optimize and self-regulate its network performance and quality of service (QoS). SDN also allows for software-defined security (SDS). SDN is ideal for today's organizations to reduce the cost and complexity of their networks while improving efficiency and security, and SDN is particularly attractive for operators as they move to 4.5G and 5G for advanced services such as IoT and HPC applications.

SDN makes the network directly programmable through the decoupling of the data and control planes to abstract the network infrastructure for network applications and services. Throughout the SDN, the configurations are updated centrally or on a particular controller instead of modifying individual network device configuration for a simple change in the system. As the data plane is separated from the control plane, the control plane can dynamically handle multiple devices and instructs the data plane to transmit the data to a defined destination. Figure 1 shows the architecture of SDN.



**Figure 1.** SDN Architecture.

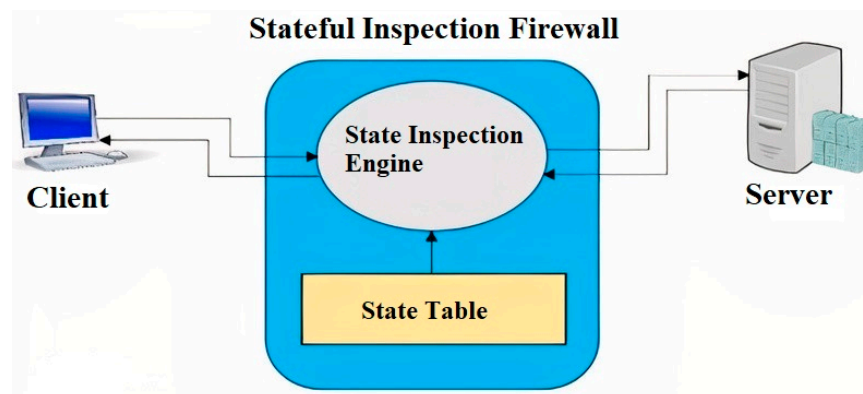
A dynamically managed network that offers flexibility, scalability, and adaptability to build a cost-effective network is possible with the help of SDN, a developing paradigm. SDN streamlines network components to enhance network performance and management for effective network administration. SDN uses OpenFlow (OF) as a communication protocol to allow the control plane and data plane to communicate with one another. Southbound interface (SBI) serves as an interface between the controller and the network component, enabling communication between the two controllers and, among other things, enabling the controller to modify the networking device's data plane forwarding tables [1]. Network elements in the same network are synchronized to send and receive update and discovery messages. The SDN controller functions as a proxy in a distributed setting or as a control agent for flows in a centralized setting under administrative control. In this manner, the network management layer, which is often accessible by a network OSS, may be fronted by the controller [2]. The SDN controller serves as a crucial administrative interface for the hosts' software switches and routers in a data center. Since they are in charge of the associated state for their transient network entities (via the agent), such as analytics and event notification, SDN controllers offer some management services in addition to provisioning and discovery.

Based on a global network perspective, the SDN controller determines a forwarding path for a specific traffic flow and then directs that forwarding path to the flow table of network elements. The received traffic flows are subsequently forwarded by switches in accordance with the flow rules established by the SDN controller. The primary function of a distributed firewall is to obstruct an attacker's horizontal mobility that is not taken into account by current stateless SDN firewalls. Network policies can be implemented concurrently and in a flexible manner thanks to distributed SDN controllers. High network availability is provided by SDN with distributed controllers, which are made up of a group of controllers and handle the entire infrastructure. If there are any changes within a controller domain, the dispersed SDN controllers share the topology information so they can update their global network status to reflect the changes [3].

The fundamental idea is to develop distributed controllers that would evenly spread the load throughout the network. In the symmetrical technique, there is no master for the SDN controllers. Each controller carries out a certain function and interacts directly with a number of other controllers to configure the network. The number of routing path requests that can be processed by an SDN controller is limited, which limits network performance. This problem is addressed by SDN with distributed controllers, which deploys a number of controllers to jointly operate the network. Network performance will be enhanced by the presence of many SDN controllers since they may distribute the network load among them. The ability of the SDN controller to control various flows from switches to destination

routes is represented by the term scalability. To improve network stability, when one controller fails, another one will take over [4].

In the centralized architecture, the dependability of SDN controllers can significantly decline. The status of the packet flow is not retained. Threats present in the SDN data plane are difficult to identify in the absence of packet status information. Before the advent of stateful firewalls, firewalls are stateless. Every traffic packet entering the network is separately inspected by a stateless firewall. Stateless firewall packet filtering does not take into account the nature of traffic patterns. As a result, this kind of firewall does not examine network packets or check the status of the connection before validating and allowing a packet into the network. State-conscious firewalls are now thought to offer improved network security and traffic control [5]. Figure 2 shows the stateful inspection firewall.



**Figure 2.** Stateful inspection firewall.

In stateful inspection firewalls, the state table is kept up-to-date to manage connections across network applications, and the packet inspection functionality enables some new traffic flows to intelligently connect to pre-existing connections.

## 2. Problem Description

SDN has a promising future in networking. SDN is developing as a powerful network management architecture for enterprise networks like datacenters, network service providers and cloud service providers. SDN tends to play a major role in solving conventional network issues. The network topology is comprised of several integrated routers, switches and firewalls that cannot be conveniently managed by a centralized hierarchical controller. SDN is capable of providing reliability, scalability, accessibility, and avoids single point of failure problems by using a logically centralized controller environment. Firewalls offer vital security both for information and business processes. Firewalls are software that track and control the traffic flows in and out of the network, filtered as per pre-configured filtering rules.

To provide robust security in the SDN framework, stateful inspection is required. The stateful firewall is responsible for tracking and identifying network traffic states based on network behaviors and flows, to monitor and secure the network. A major drawback of a single controller SDN model with stateful inspection is that the controller gathers a massive amount of state flow information which can transform into a bottleneck for the network. Therefore, there is a need for flow scheduling in multiple SDN controllers with the stateful inspection to provide robust security. It makes the distributed stateful firewall as a scalable and secure network framework.

The major accomplishments of this paper are:

- Stateful inspection firewall application programmed POX (an open-source python-based SDN controller with OpenFlow interface which can reuse components to discover the best path in any given topology) is implemented which maintains track of every packet's link information.

- To construct a distributed stateful firewall-enabled SDN controller which monitors the attacker's lateral movement in the network and applies flow-based security measures to avoid the propagation of network attack.
- A flow-based scheduling module is programmed in distributed SDN controllers to enhance the network scalability by dynamically differentiating the packet flows as large flow and small flow.
- An experimental SDN environment is built on a mininet emulator which uses a single command to instantly build a realistic virtual network running genuine kernel, switch, and application code on a single system (VM, cloud, or native) to setup a network cluster with three controllers experimented with tree independent topology which has 5, 10 and 15 switches.
- To measure and analyze the performance of the distributed stateful firewall-enabled SDN utilizing metrics such as network throughput, delay and network overhead.

### 3. Literature Work

A stateful firewall-enabled SDN controlled network serves as a middlebox. The controller is designed to provide stability and durability with a failure prediction and recovery system, and to improve the network performance. In the SDN architecture, the stateful firewall secures the network by monitoring the links established and maintains track of the state information [1]. SDN performance is analyzed by examining the network DoS threats and their impact on network bandwidth, using POX as the controller and mininet as the emulator by using TCP and UDP as traffic patterns [2]. Several SDN controllers focus on building a hierarchical adaptive hybrid cluster with flexible load-balancing to request a large-scale SDN shift [3]. A taxonomy of SDN access models is given as an expansion of the ONOS controller by expanding the northbound interface [NBI] in SDN architecture. SDN access model control features are utilized to provide control over distributed network infrastructure services [4]. FlowGuard is proposed to address SDN datacenter firewall issues such as uncertain flow route calculation and weak scalability; it provides a solution for network threats by studying the mechanism of the attack [5].

The FlowKeeper framework is proposed to eliminate security risks in the SDN data plane by enforcing control over the switch port to minimize the control plane load. FlowKeeper contributes to constructing a reliable data and stable control plane towards various attacks [6]. The controller inspects the stateful data plane because the control plane does not normally monitor the flow state change, which could result in conflicting local flow states within the switches, and also the variances between the switch and the controller [7]. Deep packet inspection gains the benefits of a programmable stateful SDN to minimize the processing load in packet filtering. The recommended computational power required by DPI and the link bandwidth for internal switches and DPI can be lowered collectively [8]. StateSec tracks packets that match source and destination node socket data without using the controller to identify and counteract DDoS threats dependent on in-switch computing capability [9]. The survey discusses securing SDN from DDOS attacks by limiting the controller packet flow rate. To reduce network time-outs and achieve enhanced access control, identical traffic flows are classified to detect DDoS attack packets [10].

The distributed controller enhances the robustness and flexibility of the control plane and reduces the effects of network partition problems [11]. A topology-conscious-specific firewall processing sends just the required firewall setup rules to reduce latency and control overhead, accounting traffic flows and network topology [12]. This study proposes to build an OpenFlow-based firewall program over the SDN controller; the flow table rules are preconfigured to directly manage incoming flows without the support of dedicated hardware [13]. An extended state machine is utilized in the stateful data plane processing as a superset to process actions in the OpenFlow table to achieve efficiency and reliability [14]. Many security policies, such as DROP, REDIRECT, QUARANTINE, can be applied by software applications built in FRESKO scripts to respond to network attacks by simply setting a parameter for action [15].



A flow-based distributed firewall model is trialed on an emulated network by designing around the features of OpenFlow, an open SDN standard [16]. Once network states are modified, FlowGuard examines network flow path fields to identify firewall protocol breaches and implements automated real-time breach solutions [17]. Using the Ostinato traffic generator tool, traffic size ranging from 64 bytes to 8950 bytes is created which is capable of achieving a maximum bandwidth of 9 Gbps [18]. Three major issues in network administration are allowing rapid updates in network status, ensuring assistance in a high-level language for network setup, and offering improved access and control over functions for network monitoring and configuring [19]. The switch extracts one out of every  $k$  packet from every input port for packet sampling. The header encapsulated information of the packet sample is transferred automatically to a controller repository. The controller determines the variety of flow statistics from obtained packet samples [20]. With the case of multiple SDN controllers, network performance can be improved as the controllers can share the network load. The flat controller model in distributed SDN is perfect for managing network fault tolerance. Thus, flat and leaderless designs of distributed controllers can be used to create a stable and reliable network [21].

In order to identify the optimum observation vector of the evolutionary state, which most accurately captures the state change of the evolving social network, OEQA reconstructs the genetic algorithm influenced by quantum mechanics; to assess state change degrees and identify abnormal changes to report anomalies SeaDM integrates ESCA and OEQA [22]. The caching strategy for D2D networks with numerous robot helper-aided caching moves the robot helpers to the best spots to increase system performance. With the help of the partitioned adaptive particle swarm optimization algorithm, the best places for the robot assistants can be discovered [23]. Cobweb-based redundant through-silicon-via provides a solution for the most common network failure scenarios, that uses a compound Poisson distribution defect model to fix cluster damage by designing active hardware with a high repair rate [24]. On two real-world datasets, the efficacy and reliability of ESTNet are demonstrated by modelling residual networks and stacking successive 3DCon to capture nonlinear and complex connections [25]. KeyListener is an exploit that uses a smartphone's audio hardware to infer keystrokes from touch screens and QWERTY keyboards. To minimize errors in acoustic signal attenuation-based keystroke localization, it monitors finger motions during inputs using phase change and the Doppler Effect. Security policies are developed using a context-aware binary tree-based search technique to increase keystroke correctness [26].

Numerical findings inputs are utilized to analyze the Markovian arrival process (MAP) applied to a MEC system in order to assess the effects of the offload rate on response performance and energy efficiency, as well as the effects of a VM's repair and service rates on availability, latency and energy [27]. In a semi-supervised hybrid learning environment, hybrid PU-learning-based spammer detection (hPSD) for spammer identification permits the construction of classifiers. By injecting a variety of positive data to locate the hidden spammers, it may iteratively detect multi-type spammers [28]. Target monitoring is accomplished by combining the marine integrated communication network system (MICN system) and UWSNs, which has resulted in higher service levels for a variety of applications, including data transmission, monitoring, and communication in the maritime environment [29]. Access control methods, Q-learning-based routing, and related open risks are addressed by the magnetic induction (MI)-assisted wireless powered subsurface sensor network (MI-WPUSN), which also addresses major dependability issues [30]. The challenges of complex cognition, making judgments in a high-dimensional action space, and self-adapting to system dynamics are discussed in the context of AI-powered MN design, a deep learning strategy that integrates cognition to make wiser judgments to ensure QoS and directly links the state of an MN to perceived QoS [31].

The client nodes' generated flow size is taken into account by the load balancing algorithm in use. Additionally, flows are categorized according to the dynamic flow size threshold value. By contrasting the performance of two load balancing algorithms with

distributed controllers' architecture, namely the flow-based load balancing algorithm and the traffic pattern-based load balancing method. The network's availability, manageability, and scalability are all improved by a distributed SDN controller, which gets rid of the drawbacks of a centralized SDN controller design [32]. In the SDN network, stateful firewall services are set up as VNFs to provide security and increase network scalability. The responsibility of the SDN controller is to create a set of standards and regulations to prevent dangerous network connectivity. These techniques cannot provide sufficient protection against intruder attacks that use multiple socket addresses [33]. hSDN models in the control and data planes, as well as control plane placement and scalability management issues. Another field under investigation is linked security and privacy concerns, as well as current threats and weaknesses. The analysis is then assisted by examining recent security modules and network security systems, as well as investigating potential risk detection and mitigation methodologies [34]. The POX controller uses OVS to connect with LRS just as it would with any other host. The packet forwarder system allows the controller to react to OpenFlow events like PacketIn and PortStatus by listening. The path finder module calculates the best routes via the network using the shortest path first algorithm and the LLDP module [35].

A migration strategy uses a balanced communication overhead and migration overhead. The reinforcement learning approach is used to create a migration plan that maximizes cumulative revenue while balancing communication overhead and migration overhead [36]. A policy-based routing algorithm is used in the network infrastructure to utilize the free IP addresses in the free IP pool. Due to the small flow table size in the SDN network, the technique provides optimal use of the available space in the flow table [37]. The most crucial step in the development of an expert system is knowledge acquisition. Working out the rule set and the accompanying membership values for such fuzzy system is a challenging and time-consuming task for the consultants since the potential range of if-then rules of the fuzzy system will increase exponentially with the rise in the range of input [38]. The nano-devices can be connected to the Internet due to a software module that transforms data formats and protocols between regular network domains and nano-network domains. A prototype of the module is created, and the performance of the suggested algorithm is assessed using the single tenant and multitenant communication situations [39].

To create a small-size network subgraph, state information from prior optimization rounds is used. The resultant subgraph is then searched for a more advantageous solution than the Lagrangian relaxation-based strategy [40]. Implementing information traffic utilizing optimal course determination and directing in heterogeneous vehicular organizations using a SDN-assisted approach in cases like a blockage in vehicle ad hoc networks (VANETs). The practical traffic reproduction model is put forth, and the trials produced results that improved the standard information flow structure in terms of Round-Trip Time (RTT) and Packet Delivery Ratio (PDR) [41]. A Network State Base Cusp model differentiates between unstable and confusable instances. Unstable instances can be found using a technique called Unstable Instance Detection (UID). The evaluation findings show that LICENSE can increase the overall detection performance by lowering the number of unstable instances and increasing the detection precision of unstable instances [42,43]. The inner mapping combiner (IMapC) assisted in lowering the volume of intermediate results sent back and forth to the Redis instances by locally aggregating partial outcome within the network [44].

These sections are divided by subheadings, to provide a concise and precise description of the experimental results, their interpretation, as well as the experimental conclusions.

#### 4. Proposed Method

In stateless firewall filtering, it is an easy process to spoof or create fake traffic packets which can go through a network. But stateful inspection offers complete control over packets that pass through the network as it keeps a record of all links and their state information. The traffic might consist of a new packet, part of packet with the connection

being established, linked to the previous packet or falsified packet. Depending on header field data such as source and destination IP and MAC addresses, port and sequence numbers, the state of the packet can be managed.

There are three major states in the TCP connections, link establishment, connection utilization, and link termination. A state table is typically used by the firewall to monitor the bidirectional connection across hosts and restrict packets which differ from its intended state. OpenFlow network communication protocol is used to enable distributed controllers to communicate the data plane elements to manage OpenFlow switches table entries for forwarding. Three network scenarios are built using tree topology, respectively, with single, two and three controllers and the number of switches vary as 5, 10 and 15. Scenario 1 has a single centralized stateful firewall-enabled controller to manage varying servers, hosts and switches. Scenario 2 has two stateful firewall-enabled distributed controllers to manage varying servers, hosts and switches. Scenario 3 has three stateful firewall-enabled distributed controllers to manage varying numbers of servers, hosts and switches. As the number of host machines ranges from four to twelve, number of switches ranges from five to fifteen and number of servers ranges from two to six.

In all three controller scenarios, the stateful firewall application is logically centralized on the proposed architecture. The stateful firewall inspects every incoming packet into the network and drops the packet which fails to pass the stateful firewall filtering rule and remaining packets are allowed into the network. Whenever there is a change in the firewall rule and state table information, the update messages are synchronized between the controllers. The advantages of a stateful firewall-enabled controller environment can benefit a network with several switches, where each switch will monitor the activities of directly linked hosts locally. An OpenFlow switch manages a listener module that maintains a record of stateful connection outcomes. Thus, the local flows are monitored based on flow rules and can detect fake flows based on three-way handshake connection information. When flows are monitored based on flow rules, it is noticed that the fake connection created by client hosts sends *SYN* packets first and the server sends *SYN-ACK* as a reply. Instead of sending *ACK* as a reply, the attacker client transmits data which leads to a full TCP connection. In Equation (1)  $T_f$  represents the threshold value to detect the fake connection.

$$T_f \leq \frac{pkt [state = +new, n]}{pkt [state = +estd, n]} \quad (1)$$

If the threshold value is exceeded then the event tracker detects and updates the flow table. Then the connection request packet sent from the attacker client is forwarded to the controller. When this process happens repeatedly the control plane gets populated with connection request messages and it leads to an increase in controller load. Thus, to detect and eliminate such attacks, stateful analysis and inspection is enabled in the controller.

Packets in and out of the FTP and web servers were captured and monitored using the Wireshark packet sniffing tool. A three-way handshake by TCP traffic was captured in FTP and web servers during the FTP and web sessions. Fake sessions were created and sent using an Ostinato Traffic Generator from the client machines. The created duplicate sessions acted as already established with [*SYN* = 0; *ACK* = 1] and stateful inspection detected that the source port number of the fake session completely differed from the established current session. The fake traffic created and sent could not be detected while monitoring the FTP and web servers using the Wireshark tool. The stateful inspection firewall blocked the duplicate packets by verifying the state table information. POX [45] was configured as a centralized and distributed controller. The key reason to select POX controller was that it is a simple lightweight controller which requires minimal hardware resources for installation. The POX controller can handle one or more diverse applications like firewall and load balancing centrally.

The response time of the POX controller is less compared to other existing python-based controllers. The POX controller implemented in an emulated tool is compatible with a real-time environment, the controller configuration is set up in an emulation tool and



can be used in real hardware with minimal changes in the code. The mininet emulator was used to emulate the SDN network and evaluate the performance of the network, File Transfer Protocol (FTP) traffic and web traffic was generated as network transfer data inside the emulated network. This work compares and analyzes the network performances of the three network scenarios. A single centralized stateful firewall-enabled SDN controller network is compared against distributed stateful firewall-enabled SDN with two and three controllers.

In the proposed SDN architecture the topologies were implemented and examined one by one, independently. In the first scenario, the single centralized controller setup was deployed with topology 1 (four hosts, five switches and two servers), topology 2 (eight hosts, ten switches and four servers) and topology 3 (twelve hosts, fifteen switches and six servers). In the second scenario, two distributed controllers were deployed with similar topologies (1, 2 and 3). In the third scenario, three distributed controllers were deployed with three similar topologies (1, 2 and 3).

#### 4.1. Openflow Protocol

OpenFlow is the most prominent protocol that enables SDN functionality to be integrated into the network of the datacenter.

OpenFlow [46] switches have flow tables that direct traffic across devices on the data plane. Each OpenFlow switch operates based on its flow table  $T$ , where the stateful inspection firewall can define traffic flow rules as  $R_1$  (rule 1) to  $R_n$  (rule  $n$ ).

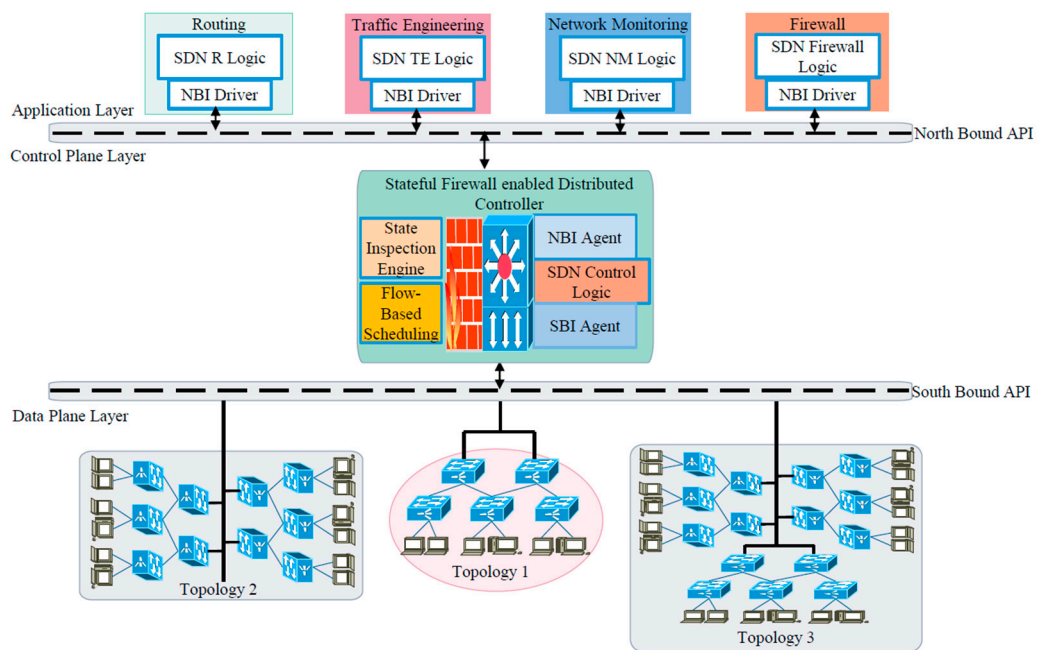
Each flow rule defined is comprised of rule priority  $R_p$ , transport layer protocol information  $P$ , whether it is TCP or UDP or FTP. Header field information  $H$  includes transport header to data layer header, rule action set information  $A_s$ , and flow statistics information  $F_s$ .

Thus, the flow rule  $R$  in switches is defined as  $R = \{H, P, A_s, R_p, F_s\}$ . The flow rule statistics  $F_s$  consists of information about each packet's flow length  $L$  and packet size  $S_p$ ,  $F_s = \{L; S_p\}$ . The header field  $H$  of the rule table comprises the source and destination physical address ( $S_p, D_p$ ), Source IP address and destination IP address ( $S_i, D_i$ ), and source and destination port number ( $S_p, D_p$ ) and port number of incoming packet's physical port  $I_p$ . Thus, the header field of a packet can be defined as  $H = \{S_p, D_p; S_i, D_i; S_p, D_p; I_p\}$ .

#### 4.2. Scenario 1: Centralized Stateful Firewall-Enabled Single Controller SDN

In Scenario 1 a single centralized POX controller which was tested independently with three different tree topologies. Topology 1 consisted of five switches, among them two switches were connected to the controller, two servers and four hosts, Topology 2 consisted of 10 switches, among them four switches were connected to the controller, four servers and eight hosts. Topology 3 consisted of 15 switches, among them six switches were connected to the controller, six servers and 12 hosts, respectively. All three-network setups were configured to handle FTP and web traffic flows.

Topology 1 was created with five switches, among them two switches were connected to the controller, one web server, one FTP server and remaining four hosts were configured as the client machine to generate and handle FTP and HTTP packets. Topology 2 was created with two web servers, two FTP servers and the remaining 8 hosts were configured as the client machine to generate and handle FTP and HTTP packets. Topology 3 was created with three web servers, three FTP servers and the remaining 12 hosts were configured as the client machine to generate and handle FTP and HTTP packets. The Ostinato traffic generator was configured in the client machine to generate web and FTP packets locally. The Wireshark packet sniffing tool was configured in web and FTP servers to monitor traffic flows. Openflow switches were used to connect clients and servers by exchanging the connection information. All three topologies were implemented separately, and the SDN controllers handled and controlled the traffic of the respective topologies individually. The Scenario 1 experimental setup was tested with three separate topologies and a summarized view of the topologies is shown in Figure 3.



**Figure 3.** Stateful firewall-enabled single centralized SDN controller.

The stateful inspection firewall algorithm was proposed to receive and inspect the ingress and egress packets and make a decision whether to permit or drop the packet into the network. The stateful inspection firewall was enabled in POX controller and the steps in Algorithm 1 are described below

---

**Algorithm 1:** Algorithm for centralized stateful packet filtering

---

Input:  $R$  states firewall rule,  $H$  states Packet Header,  $S_T$  states State Table,  $S_r$  states server and  $R_T$  states Firewall Rule Table

- 1: Start Network
  - 2: Check Network Connectivity;
  - 3: Set Firewall Rule  $\leftarrow$  rule =  $R.add(FTP \ \&\& \ HTTP \ == \ allow)$ ;
  - 4: Receive the packet;
  - 5: Network Self-Test  $\leftarrow$  mac =  $H.find(mac)$ ;
  - 6: If ( $mac.find() == 0:0:0:0$ ) then;
  - 7: Drop packet;
  - 8: Else
  - 9: Inspect State Table traffic flow  $\leftarrow$  flow =  $S_T.find(flow)$ ;
  - 10: If ( $pckt.find() == flow$ ) then;
  - 11:  $pckt.send(S_r)$ ;
  - 12: Else
  - 13: Apply Firewall Filtering  $\leftarrow$  rule =  $R_T.match(rule)$ ;
  - 14: If ( $rule.match() == allow$ ) then;
  - 15:  $pckt.send(S_r)$ ;
  - 16: Else
  - 17: Drop packet;
  - 18: Endif
  - 19: Endif
  - 20: Endif
  - 21: END
- 

Steps 1 to 7: Start the POX controller and check the connectivity between OpenFlow switches and controller. Set a rule on the controller to allow client machines to access FTP

and Web servers. Initial network sanity test is completed. The traffic flow is allowed into the stateful firewall filtering procedure when it passes the sanity check or drops the packet.

Steps 8 to 11: State information of the incoming packet is analyzed and the packet is sent to a particular server if a match is identified in the state table.

Steps 12 to 21: The received packet is subjected to stateful filtering. If a rule match is found then the packet information is stored on the state table. If not drop the packet.

All the network traffic flows were directed to a centralized SDN controller for stateful packet inspection and packet state entries were added to the state table. This process can rapidly overload the centralized SDN controller. If the network link monitoring is activated on centralized SDN controller the intruder could initiate attack programs which can saturate and overload the control plane. As a result, a distributed controller environment is required for datacenter SDN.

4.3. Scenario 2: Distributed Stateful Firewall Enabled SDN with Two Controllers

Figure 4 shows the Scenario 2 experimental setup.

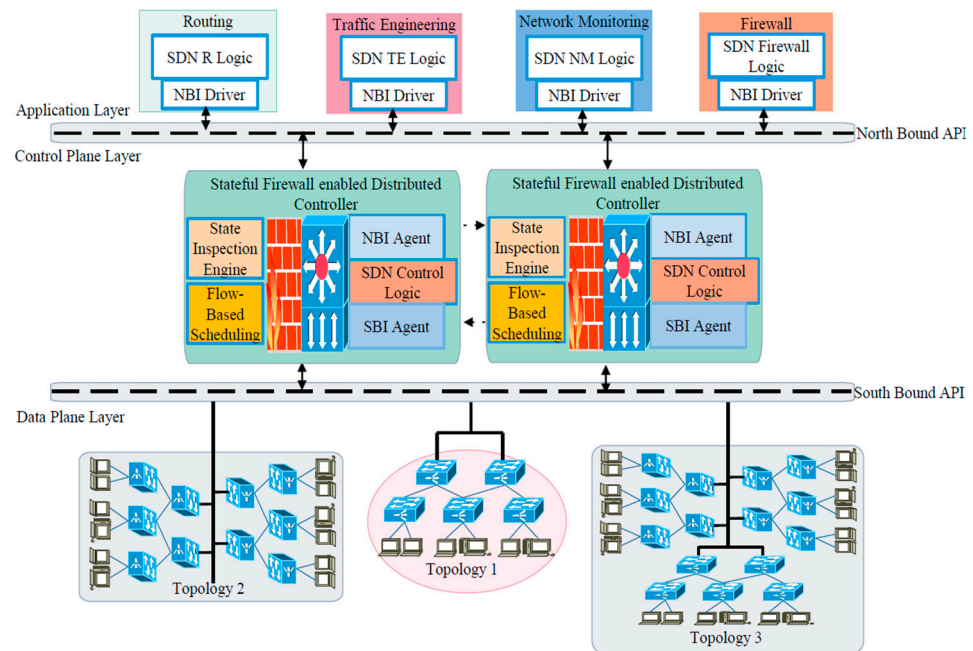


Figure 4. Stateful firewall-enabled two distributed SDN controller.

It was created with two distributed POX controllers which were tested independently with three different tree topologies. The rest of the experimental setup parameters which include network topologies, traffic generation and network monitoring were identical to scenario 1.

4.4. Scenario 3: Distributed Stateful Firewall Enabled SDN with Three Controllers

Figure 5 shows the Scenario 3 experimental setup. It was created with three distributed POX controllers which were tested independently with three different tree topologies. The rest of the experimental setup parameters such as network topologies, traffic generation and network monitoring were identical to scenario 1 and 2.

The distributed stateful firewall inspection algorithm was designed to receive a packet and keep track of the connection between controllers and OpenFlow switches. Stateful firewall inspection was enabled in both the controllers to filter malicious traffic into the network.

Steps 1 to 3: Start all the POX controllers in parallel and check the connectivity between the distributed controllers and OpenFlow switches and receive the incoming packets and requests into the network.

Steps 4 to 8: The incoming packets and their connection status are examined. If the examined connection is active at that moment the corresponding packet is sent to the state table module to process and update packet status in the state table.

Steps 9 to 18: The packet is sent to event tracker and subjected to stateful filtering. If the packet matches the permit rule, the packet information is forwarded to the OpenFlow table, or else the packet gets dropped.

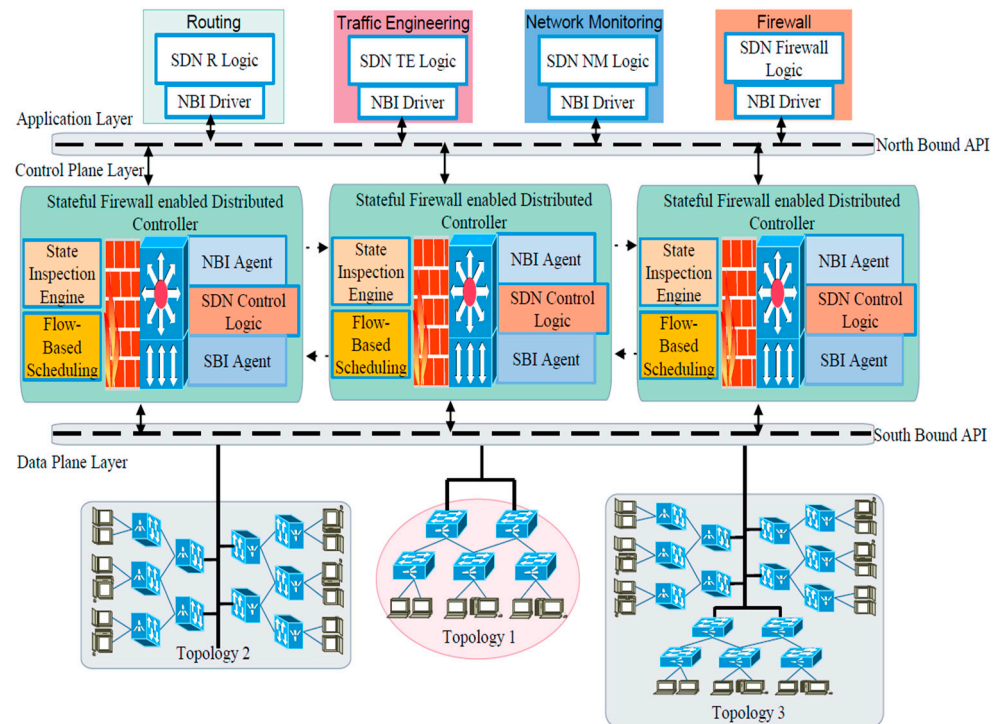


Figure 5. Stateful firewall enabled three distributed SDN controller.

Figure 6 shows the flow for Algorithm 2.

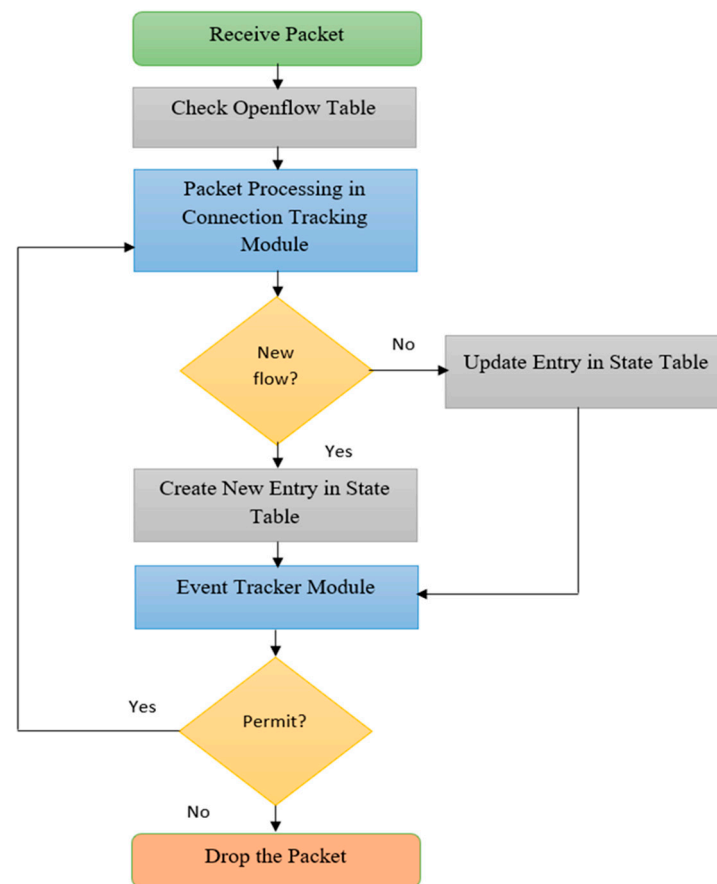
---

**Algorithm 2:** Design for Distributed Stateful Inspection

---

*Input:*  $OFT$  states Openflow Table,  $STM$  states State Table Module,  $S_T$  states State Table and  $E_T$  states Event Tracker

- 1: Start
  - 2: Check Network Connectivity;
  - 3: Receive Packet;
  - 4: Examine Openflow Table to trace connection  $\leftarrow pkt = OFT.find(pkt)$ ;
  - 5: If  $(pkt.find() = active)$  then;
  - 6:  $pkt.send(STM)$ ;
  - 7: Update State Table  $\leftarrow pkt = S_T.update(pkt)$ ;
  - 8:  $pkt.send(E_T)$ ;
  - 9: Else
  - 10:  $pkt.send(E_T)$ ;
  - 11: Inspect and Validate the Packet  $\leftarrow rule = E_T.match(rule)$ ;
  - 12: If  $(rule.match() = allow)$  then;
  - 13:  $pkt.send(OFT)$ ;
  - 14: Else
  - 15: Drop Packet;
  - 16: Endif
  - 17: Endif
  - 18: END
-



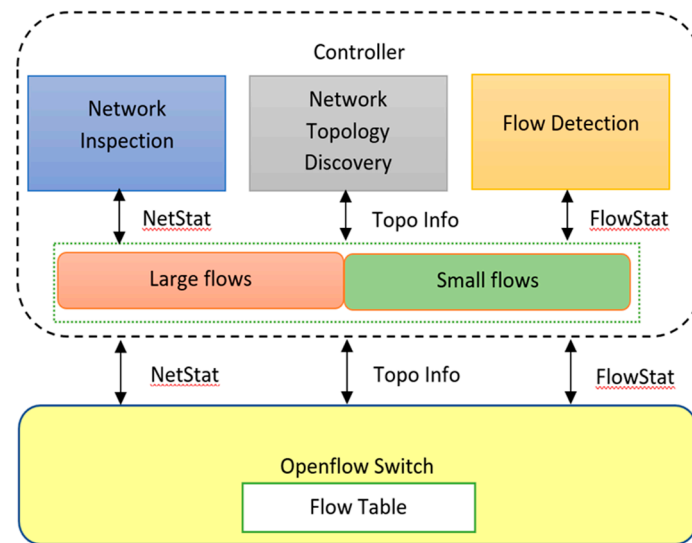
**Figure 6.** Design of stateful inspection algorithm.

A fake packet is generated from a client machine and sent to the web or FTP server through the controller. The created connection is tracked and sent to the event tracker module. The event tracker verifies the connection information and finds that the packet received is fake and drops the packet. The scalability and reliability of the distributed controller network are better as they share the incoming traffic between them and prevent a network bottleneck problem. The distributed SDN controllers synchronize with each other to update changes in the network and are logically centralized to efficiently manage data plane elements. The controller updates whenever the topology changes, a new switch is added or an existing switch is removed, a new link is configured or an existing link is removed or there is a change in network policy. In the distributed controller environment when a controller detects a new element or removal of an existing element, or when a controller goes down, a notification message is sent to other controllers instantly. Hence the controller load is reduced in the distributed stateful firewall, which filters the duplicate connection much efficiently than a centralized stateful inspection with increased performance.

#### 4.5. Flow-Based Scheduling Module

The flow-based scheduling module shown in Figure 7 is configured in the stateful firewall-enabled distributed controller which manages and schedules control flows by differentiating the traffic pattern as large and small flows and dynamically allocating a path based on network load information.





**Figure 7.** Flow-based scheduling.

#### Steps in flow-based scheduling module

1. Network topology discovery: This sub-module is configured in the controller to acquire link status information between OpenFlow switches and controllers, so that it can maintain an overall view of the network topology.
2. Network Inspection: is responsible for regular monitoring the connection information between OpenFlow switches, distributed controllers and link utilization status.
3. Flow detection: classifies the current flows as large flow or small flow by continuous comparing and analyzing of incoming flows with the pre-configured flow-based scheduling module. Initially there is no prior knowledge of flows on the flow-based scheduling module. The flows are classified based on file size distribution across the network and flow completion time. When a communication is established initially there will quick exchange of packets which are marked as small flows and an extensive transmission of data which are marked as large flows. When a flow is identified as large flow it is sent to a large flow sub-module or the flow is sent to a small flow sub-module. From there, the paths for respective flows are assigned with a view to the network link load information.

The paths for large flows and small flows are classified and paths for the respective flow selected by calculating every probable shortest path with maximum bandwidth. When all probable shortest paths are found, the best shortest path ( $Flow_{sp}$ ) is selected as mentioned in Equation (2).

$$Flow_{sp} = \frac{[Flow_{rate} + Link_{load}]}{Link_{Bandwidth}} \quad (2)$$

Thus, the proposed flow-based scheduling module helps in minimizing overall network response and reduces the overhead of the network to improve network scalability and performance. The proposed work has both stateful packet inspection and flow-based scheduling features with distributed controllers. When the flow reaches the controller, it is filtered and classified to route the packet to its destination. The header field of the packet is analyzed to record network connectivity details such as IP addresses of source and destination, port addresses of source and destination, current session details and packet sequence identifier. By analyzing the incoming packets, the stateful firewall tracks the traffic and their state table details. If the state table contains no current link information, then the packet is subjected to stateful packet inspection. When the packet is allowed by the firewall rule it is permitted in and out of the network and the firewall events are recorded in the session table. If the packet fails the firewall filtering rule, the respective packet gets

dropped. Thus, a stateful firewall application is designed to block malicious packets by functioning as an intrusion prevention system.

## 5. Implementation Results and Discussion

This segment discusses the performance outcome of the proposed method as well as the descriptions of the network setup in Table 1. The network performance was evaluated using web and file transfer traffic flows. The stateful inspection firewall application was programmed to filter the packets in and out of the distributed SDN controller. The incoming FTP and HTTP traffic packets were subjected to stateful inspection to measure and evaluate the values of the network throughput, delay, and network overhead. In all the controller scenarios, the incoming traffic was subjected to the stateful packet filtering rule to eliminate undesirable flows into the network. Then the permitted traffic was classified based on the flow type, which led to minimizing the controller-to-traffic load. This improved the overall performance of the network concerning network throughput, delay and network overhead. The analysis demonstrated that scenario 3 is an ideal SDN model compared to scenarios 1 and 2. All output parameters were calculated through several simulation iterations and the assessed results are presented as average.

**Table 1.** Simulation setup.

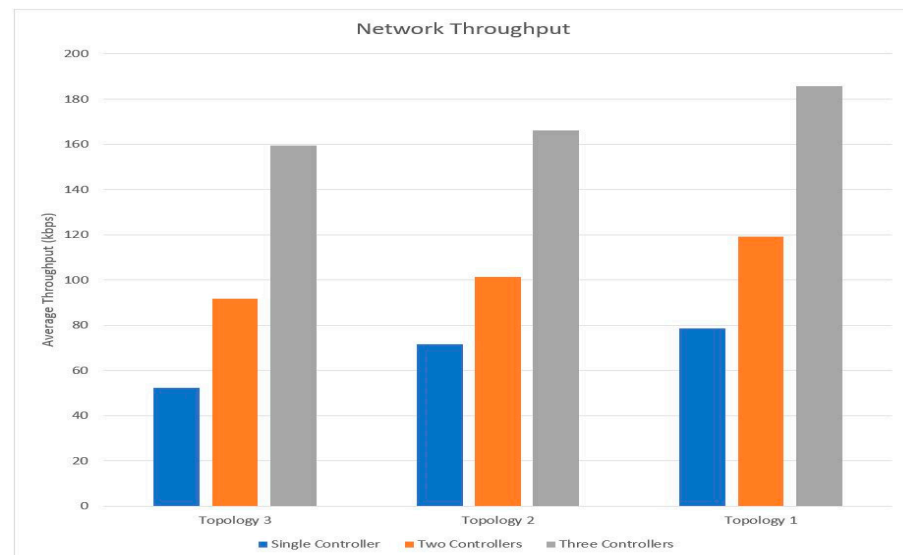
| Hardware and Software         | Description                    |
|-------------------------------|--------------------------------|
| CPU                           | Intel core i5-8250U @ 1.60 Ghz |
| Memory                        | 8 GB RAM                       |
| Operating System              | Windows 10                     |
| Virtualization Tool           | Oracle Virtual Box-5.1         |
| Virtualized OS                | Ubuntu (64 bit) 14.04          |
| Virtual Machine configuration | 512 MB memory, 1 CPU core      |
| Mininet tool version          | 2.2.1                          |
| POX controller version        | 0.2.0 (carp)                   |
| Programming Language version  | 2.7                            |
| Interface Protocol            | OF 1.0                         |

### 5.1. Network Throughput

The network throughput value is determined in a network by examining the number of packets handled by the control plane per seconds. Iperf was used to measure the throughput of stateful firewall enabled SDN controller using parameters like TCP window size and Round-Trip Time. Network throughput performance was measured based on a single centralized POX controller, two distributed POX controllers and three distributed controllers with three independent topologies. In the centralized controller scenario, there is one controller which has to take care of firewall application and flow scheduling activities, which increases the load of the controller and forces the controller to handle more traffic. As the control and management packets of the controller share the connectivity resource the incoming flow handling capability of the controller degrades due to excessive and unexpected load.

Thus, the throughput of the centralized controller is low compared with other two proposed scenarios. In the distributed controller scenario with two controllers, the stateful firewall performs stateful packet filtering and flows were classified using the flow-based scheduling module on both the controllers. Even though the management and control traffic shared the same links for communication, the distributed controllers shared the overall traffic load among them evenly. Thus, the network throughput of scenario two is better compared to scenario one with a single centralized controller. In scenario 3 with three distributed controllers the overall controller load was distributed evenly among three

controllers and the network throughput performance was enhanced as the link was readily available for the packet transfer from source to destination. Figure 8 shows the calculated difference, in network throughput values. When comparing SDN with two distributed controllers against a single centralized controller, the network throughput increases by 76% in topology 1, 42% in topology 2 and 53% in topology 3. When comparing the SDN with three distributed controllers against SDN with two distributed controllers, network throughput increases by 73% in topology 1, 64% in topology 2 and 55% in topology 3.



**Figure 8.** Network Throughput.

### 5.2. Network Delay

The network delay is measured by estimating the time taken by the network to send, process and receive the packet in milliseconds. Average end-to-end delay network delay performance values were measured based on a single centralized POX controller, two distributed POX controllers and three distributed controllers with three independent topologies. As the packet enters the network it is subjected to the stateful firewall filtering rule and if the packet is allowed into the network a new state information is created and stored in the state table. Thus, by recording the active connection entries in the state table the delay of the network is minimized because there is no need for the controller to inspect the flow until the established connection gets terminated. Then the flow-based scheduling module helps in the pathfinding process by considering the size of the flow; when a packet is permitted by the firewall module the flow is analyzed and classified according to its size to allocate the best path to the destination. The flow-based scheduling network topology discovery sub-module has a global view of the network and the network inspection has all the link status information.

Thus, the response time of a controller is minimized and major delay caused by packet inspection and pathfinding is reduced. The delay of the distributed SDN with three controllers is very minimal compared to a distributed SDN with two controllers and centralized controller scenarios because the capability of the controller is improved to handle a greater number of packet requests. When the network size increases the distributed controller can control and manage a packet request efficiently as the controllers synchronize and update topology information whenever there is a change in the network topology. Figure 9 shows the calculated difference in delay values. When comparing SDN with two distributed controllers against a single centralized controller, the delay decreases by 51% in topology 1, 55% in topology 2 and 56% in topology 3. When comparing the SDN with three distributed controllers against the SDN with two distributed controllers, the delay decreases by 35% in topology 1, 42% in topology 2 and 53% in topology 3.

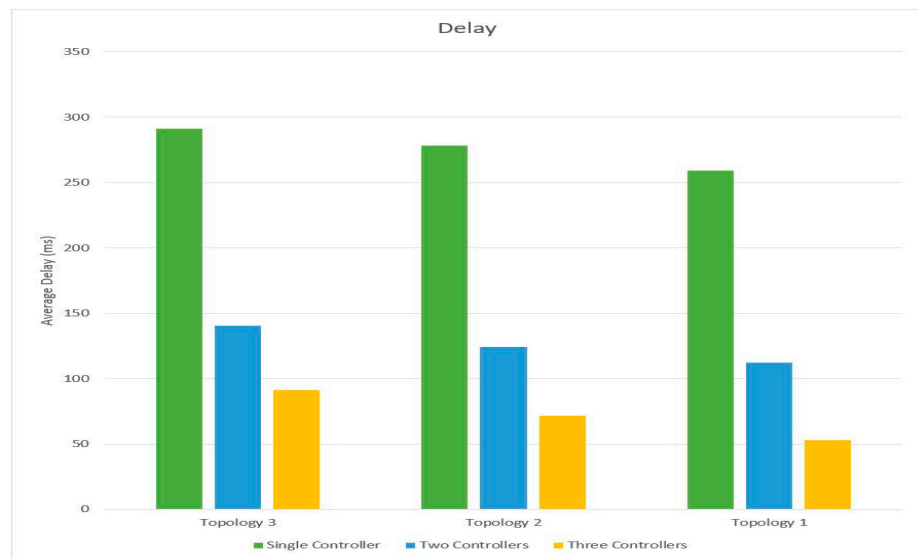


Figure 9. Network delay.

### 5.3. Network Overhead

The network overhead is estimated by measuring the number of packets or flows handled by the control plane at a specific time. The unit for measuring network overhead is packets per millisecond (ppms). Average network overhead performance values were measured based on a single centralized POX controller, two distributed POX controllers and three distributed controllers with three independent topologies. Figure 10 shows the calculated difference in network overhead values. As the controller distributes the network traffic load between the controllers to minimize the amount of traffic handled by a controller, it helps the proposed architecture to reduce overall network overhead. The stateful firewall application initially inspects the incoming traffic header information and maintains active connection information in the state table. The overall controller overhead is reduced and there is no need for the controller to verify and validate all the flow requests of the active connection; as an outcome the controller can process a greater number of packets in a given time.

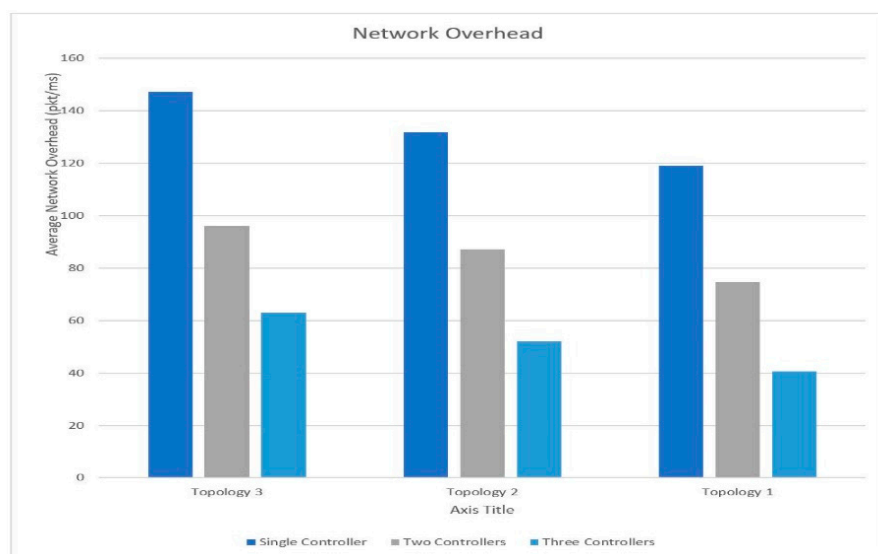


Figure 10. Network overhead.

Furthermore, the flow-based scheduling module classifies the flows requests and helps to reduce the overhead of the controller by providing network status, flow status

and topology information. The distributed SDN with three controllers performs better when comparing the distributed controller with two controllers and a single centralized controller. As the number of controllers increases, the number of packets handled at a specific time also increases and thus the overhead of the network is reduced in scenario 3 efficiently. When comparing the SDN with two distributed controllers against a single centralized controller, the overhead decreases by 35% in topology 1, 34% in topology 2 and 37% in topology 3.

When comparing the SDN with three distributed controllers against the SDN with two distributed controllers, the network overhead decreases by 33% in topology 1, 41% in topology 2 and 45% in topology 3. The firewall with stateful inspection in distributed SDN controllers secures and handles the traffic flows effectively to enhance the network performance. When comparing the overall values of average network throughput, average delay and average network overhead of a single centralized controller and distributed SDN with two controllers, the network throughput performance is increased by 57%, the delay is decreased by 54% and network overhead is reduced by 35%. By comparing a distributed SDN with two controllers and distributed SDN with three controllers the overall network throughput is increased by 64%, the delay is decreased by 43% and network overhead is reduced by 39%.

## 6. Conclusions

In this work, three stateful firewall-enabled SDN controller scenarios were created and tested with three independently working topologies. The designed experimental setup replicated a mini datacenter network managed with a single centralized controller, two distributed controllers and three distributed controllers, respectively. The network handles real-time traffic flows like web and FTP. To route the packet to its destination, the flow is filtered and categorized as it reaches the controller. The packet's header field is examined to capture information about network connectivity, including source and destination IP addresses, port addresses, details of the current session, and a packet sequence identification. The stateful firewall monitors the traffic and the specifics of the state table by examining the incoming packets. The firewall events are recorded in the session table when the firewall rule permits the packet to enter or exit the network. The packet is dropped if it violates a firewall filtering rule. A flow-based scheduling mechanism minimizes the path selection process, assures bandwidth for flows and improves scalability. The performance of the proposed network was assessed by measuring the overall values of network throughput, delay and network overhead. By comparing the performance outcomes, it was found that the stateful firewall-enabled distributed controller with three controllers proved to be a promising SDN network environment. This work can be extended in the future by increasing network size and implementing a load balancing algorithm over the SDN to efficiently manage the increasing traffic needs.

**Author Contributions:** S.P. and K.C.: research concept and methodology, writing—original draft preparation; B.P.K. and S.R.S.: investigation; R.V.: supervision and revision, W.-C.L.: validation and funding acquisition, review and editing. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research has been funded by the National Yunlin University of Science and Technology, Douliu.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

**Conflicts of Interest:** The authors declare no conflict of interest.



## References

1. Abdullah, A.F.; Salem, F.M.; Tammam, A.; Azeem, M.H.A. Performance Analysis and Evaluation of Software Defined Networking Controllers against Denial of Service Attacks. *J. Phys. Conf. Ser.* **2020**, *1447*, 012007. [[CrossRef](#)]
2. Afek, Y.; Bremner-Barr, A.; Feibish, S.L.; Schiff, L. Sampling and Large Flow Detection in SDN. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 345–346. [[CrossRef](#)]
3. Bianchi, G.; Bonola, M.; Capone, A.; Cascone, C. OpenState. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 44–51. [[CrossRef](#)]
4. Boite, J.; Nardin, P.-A.; Rebecchi, F.; Bouet, M.; Conan, V. Statesec: Stateful monitoring for DDoS protection in software defined networks. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; pp. 1–9. [[CrossRef](#)]
5. Dargahi, T.; Caponi, A.; Ambrosin, M.; Bianchi, G.; Conti, M. A Survey on the Security of Stateful SDN Data Planes. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1701–1725. [[CrossRef](#)]
6. Dayal, N.; Maity, P.; Srivastava, S.; Khondoker, R. Research Trends in Security and DDoS in SDN. *Secur. Commun. Networks* **2016**, *9*, 6386–6411. [[CrossRef](#)]
7. Dixit, V.H.; Kyung, S.; Zhao, Z.; Doupé, A.; Shoshitaishvili, Y.; Ahn, G.-J. Challenges and Preparedness of SDN-based Firewalls. In Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, Tempe, AZ, USA, 21 March 2018; pp. 33–38. [[CrossRef](#)]
8. Tran, T.V.; Ahn, H. A network topology-aware selectively distributed firewall control in SDN. In Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, 28–30 October 2015; pp. 89–94. [[CrossRef](#)]
9. Gao, S.; Li, Z.; Xiao, B.; Wei, G. Security Threats in the Data Plane of Software-Defined Networks. *IEEE Netw.* **2018**, *32*, 108–113. [[CrossRef](#)]
10. Hu, F.; Hao, Q.; Bao, K. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 2181–2206. [[CrossRef](#)]
11. Hu, H.; Han, W.; Ahn, G.-J.; Zhao, Z. FLOWGUARD: Building robust firewalls for software-defined networks. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; pp. 97–102. [[CrossRef](#)]
12. Kim, H.; Feamster, N. Improving network management with software defined networking. *IEEE Commun. Mag.* **2013**, *51*, 114–119. [[CrossRef](#)]
13. Kreutz, D.; Ramos, F.M.V.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [[CrossRef](#)]
14. Liu, Y.; He, Q.; Li, X.; Zhou, S. A Distributed Dynamic Adaptive and Fast Balancing SDN Controller Management. *IOP Conf. Series: Earth Environ. Sci.* **2019**, *234*, 012027. [[CrossRef](#)]
15. Oktian, Y.E.; Lee, S.; Lee, H.; Lam, J. Distributed SDN controller system: A survey on design choice. *Comput. Networks* **2017**, *121*, 100–111. [[CrossRef](#)]
16. Paladi, N.; Gehrmann, C. SDN Access Control for the Masses. *Comput. Secur.* **2018**, *80*, 155–172. [[CrossRef](#)]
17. Pena, J.G.V.; Yu, W.E. Development of a distributed firewall using software defined networking technology. In Proceedings of the IEEE International Conference on Information Science and Technology, Shenzhen, China, 26–28 April 2014; pp. 449–452. [[CrossRef](#)]
18. Prabakaran, S.; Ramar, R. Stateful firewall-enabled software-defined network with distributed controllers: A network performance study. *Int. J. Commun. Syst.* **2019**, *32*, e4237. [[CrossRef](#)]
19. Sanvito, D.; Moro, D.; Capone, A. Towards traffic classification offloading to stateful SDN data planes. In Proceedings of the IEEE Conference on Network Softwarization (NetSoft), Milan, Italy, 3–7 July 2017; pp. 1–4. [[CrossRef](#)]
20. Srivastava, S.; Anmulwar, S.; Sapkal, A.; Batra, T.; Gupta, A.K.; Kumar, V. Comparative study of various traffic generator tools. In Proceedings of the Recent Advances in Engineering and Computational Sciences (RAECS), Chandigarh, India, 6–8 March 2014; pp. 1–6. [[CrossRef](#)]
21. Suh, M.; Park, S.H.; Lee, B.; Yang, S. Building firewall over the software-defined network controller. In Proceedings of the International Conference on Advanced Communication Technology, PyeongChang, Korea, 16–19 February 2014; pp. 744–748. [[CrossRef](#)]
22. Wang, H.; Gao, Q.; Li, H.; Wang, H.; Yan, L.; Liu, G. A Structural Evolution-Based Anomaly Detection Method for Generalized Evolving Social Networks. *Comput. J.* **2020**, *65*, 1189–1199. [[CrossRef](#)]
23. Lin, Y.; Song, H.; Ke, F.; Yan, W.; Liu, Z.; Cai, F. Optimal caching scheme in D2D networks with multiple robot helpers. *Commun. Commun.* **2021**, *181*, 132–142. [[CrossRef](#)]
24. Ni, T.; Liu, D.; Xu, Q.; Huang, Z.; Liang, H.; Yan, A. Architecture of Cobweb-Based Redundant TSV for Clustered Faults. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 1736–1739. [[CrossRef](#)]
25. Luo, G.; Zhang, H.; Yuan, Q.; Li, J.; Wang, F.-Y. ESTNet: Embedded Spatial-Temporal Network for Modeling Traffic Flow Dynamics. *IEEE Trans. Intell. Transp. Syst.* **2022**, 1–12. [[CrossRef](#)]
26. Yu, J.; Lu, L.; Chen, Y.; Zhu, Y.; Kong, L. An Indirect Eavesdropping Attack of Keystrokes on Touch Screen through Acoustic Sensing. *IEEE Trans. Mob. Comput.* **2019**, *20*, 337–351. [[CrossRef](#)]
27. Wang, Y.; Han, X.; Jin, S. MAP based modeling method and performance study of a task offloading scheme with time-correlated traffic and VM repair in MEC systems. *Wirel. Networks* **2022**, *28*, 1–22. [[CrossRef](#)]

28. Wu, Z.; Cao, J.; Wang, Y.; Wang, Y.; Zhang, L.; Wu, J. hPSD: A Hybrid PU-Learning-Based Spammer Detection Model for Product Reviews. *IEEE Trans. Cybern.* **2018**, *50*, 1595–1606. [[CrossRef](#)]
29. Lv, Z.; Chen, D.; Feng, H.; Wei, W.; Lv, H. Artificial Intelligence in Underwater Digital Twins Sensor Networks. *ACM Trans. Sens. Networks* **2022**, *18*, 1–27. [[CrossRef](#)]
30. Liu, G. Data Collection in MI-Assisted Wireless Powered Underground Sensor Networks: Directions, Recent Advances, and Challenges. *IEEE Commun. Mag.* **2021**, *59*, 132–138. [[CrossRef](#)]
31. Luo, G.; Yuan, Q.; Li, J.; Wang, S.; Yang, F. Artificial Intelligence Powered Mobile Networks: From Cognition to Decision. *IEEE Netw.* **2022**, *36*, 136–144. [[CrossRef](#)]
32. Prabakaran, S.; Ramar, R. Software Defined Network: Load Balancing Algorithm Design and Analysis. *Int. Arab J. Inf. Technol.* **2021**, *18*, 312–318. [[CrossRef](#)]
33. Prabakaran, S.; Ramar, R.; Hussain, I.; Kavim, B.P.; Alshamrani, S.S.; AlGhamdi, A.S.; Alshehri, A. Predicting Attack Pattern via Machine Learning by Exploiting Stateful Firewall as Virtual Network Function in an SDN Network. *Sensors* **2022**, *22*, 709. [[CrossRef](#)]
34. Khorsandroo, S.; Sánchez, A.G.; Tosun, A.S.; Arco, J.; Doriguzzi-Corin, R. Hybrid SDN evolution: A comprehensive survey of the state-of-the-art. *Comput. Networks* **2021**, *192*, 107981. [[CrossRef](#)]
35. Ahmad, S.; Mir, A.H. Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers. *J. Netw. Syst. Manag.* **2020**, *29*, 9. [[CrossRef](#)]
36. Li, C.; Qianqian, C.; Luo, Y. Low-latency edge cooperation caching based on base station cooperation in SDN based MEC. *Expert Syst. Appl.* **2021**, *191*, 116252. [[CrossRef](#)]
37. Paliwal, M.; Nagwanshi, K.K. Effective Flow Table Space Management Using Policy-Based Routing Approach in Hybrid SDN Network. *IEEE Access* **2022**, *10*, 59806–59820. [[CrossRef](#)]
38. Vijay, S.A.A.; GaneshKumar, P. Fuzzy Expert System based on a Novel Hybrid Stem Cell (HSC) Algorithm for Classification of Micro Array Data. *J. Med Syst.* **2018**, *42*, 61. [[CrossRef](#)]
39. Galal, A.; Hesselbach, X.; Tavernier, W.; Colle, D. SDN-based gateway architecture for electromagnetic nano-networks. *Comput. Commun.* **2021**, *184*, 160–173. [[CrossRef](#)]
40. BinSahaq, A.; Sheltami, T.; Mahmoud, A.; Nasser, N. Fast and efficient algorithm for delay-sensitive QoS provisioning in SDN networks. *Wirel. Networks* **2022**, *28*, 1–22. [[CrossRef](#)]
41. Tao, H.; Zain, J.M.; Band, S.B.; Sundaravadivazhagan, B.; Mohamed, A.; Marhoon, H.A.; Ogbonnia, O.O.; Young, P. SDN-assisted technique for traffic control and information execution in vehicular adhoc networks. *Comput. Electr. Eng.* **2022**, *102*, 108108. [[CrossRef](#)]
42. Ran, L.; Cui, Y.; Guo, C.; Qian, Q.; Shen, G.; Xing, H. Defending saturation attacks on SDN controller: A confusable instance analysis-based algorithm. *Comput. Networks* **2022**, *213*, 109098. [[CrossRef](#)]
43. Netcharoensirisuk, P.; Abrahamian, C.; Tang, R.; Chen, C.C.; Rosato, A.S.; Beyers, W.; Chao, Y.K.; Filippini, A.; Di Pietro, S.; Bartel, K.; et al. Flavonoids increase melanin production and reduce proliferation, migration and invasion of melanoma cells by blocking endolysosomal/melanosomal TPC2. *Sci. Rep.* **2021**, *11*, 8515. [[CrossRef](#)]
44. Jhaveri, R.H.S.; Ramani, S.V.; Srivastava, G.; Gadekallu, T.R.; Aggarwal, V. Fault-Resilience for Bandwidth Management in Industrial Software-Defined Networks. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 3129–3139. [[CrossRef](#)]
45. Kavitha, C.; Srividhya, S.R.; Lai, W.-C.; Mani, V. IMapC: Inner MAPPING Combiner to Enhance the Performance of MapReduce in Hadoop. *Electronics* **2022**, *11*, 1599. [[CrossRef](#)]
46. Jemmali, M.; Denden, M.; Boulila, W.; Jhaveri, R.H.; Srivastava, G.; Gadekallu, T.R. A Novel Model Based on Window-Pass Preferences for Data-Emergency-Aware Scheduling in Computer Networks. *IEEE Trans. Ind. Inform.* **2022**, *18*, 7880–7888. [[CrossRef](#)]