




Article

Adaptive Path Selection Algorithm with Flow Classification for Software-Defined Networks

Muhammed Nura Yusuf ^{1,2,*}, Kamalrulnizam bin Abu Bakar ¹, Babangida Isyaku ^{1,3}, Ahmed Hamza Osman ⁴, Maged Nasser ⁵ and Fatin A. Elhaj ⁶

¹ Faculty of Computing, Universiti Teknologi Malaysia, Johor 81310, Malaysia

² Department of Mathematical Science, Abubakar Tafawa Balewa University, Bauchi PMB 0284, Bauchi State, Nigeria

³ Faculty of Computing and Information Technology, Sule Lamido University, Kafin Hausa PMB 047, Jigawa State, Nigeria

⁴ Department of Information System, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Jeddah 21911, Saudi Arabia

⁵ School of Computer Sciences, Universiti Sains Malaysia, Penang 11800, Malaysia

⁶ College of Art, Science and Information Technology, University of Khorfakkan, Sharjah P.O. Box 18119, United Arab Emirates

* Correspondence: nyusuf@graduate.utm.my or ymnura@atbu.edu.ng

Abstract: Software-Defined Networking (SDN) is a trending architecture that separates controller and forwarding planes. This improves network agility and efficiency. The proliferation of the Internet of Things devices has increased traffic flow volume and its heterogeneity in contemporary networks. Since SDN is a flow-driven network, it requires the corresponding rule for each flow in the flowtable. However, the traffic heterogeneity complicates the rules update operation due to varied quality of service requirements and en-route behavior. Some flows are delay-sensitive while others are long-lived with a propensity to consume network buffers, thereby inflicting congestion and delays on the network. The delay-sensitive flows must be routed through a path with minimal delay, while congestion-susceptible flows are guided along a route with adequate capacity. Although several efforts were introduced over the years to efficiently route flows based on different QoS parameters, the current path selection techniques consider either link or switch operation during decisions. Incorporating composite path metrics with flow classification during path selection decisions has not been adequately considered. This paper proposes a technique based on composite metrics with flow classification to differentiate congestion-prone flows and reroute them along appropriate paths to avoid congestion and loss. The technique is integrated into the SDN controller to guide the selection of paths suitable to each traffic class. Compared to other works, the proposed approach improved the path load ratio by 25%, throughput by 35.6%, and packet delivery ratio by 31.7%.

Keywords: SDN; path selection; path quality; heterogeneous traffic; elephant flow; mice flow

MSC: 68P20; 68P10; 63E72; 68U15



Citation: Yusuf, M.N.; Bakar, K.b.A.; Isyaku, B.; Osman, A.H.; Nasser, M.; Elhaj, F.A. Adaptive Path Selection Algorithm with Flow Classification for Software-Defined Networks. *Mathematics* **2023**, *11*, 1404. <https://doi.org/10.3390/math11061404>

Academic Editors: Andrey Koucheryavy, Ahmed A. Abd El-Latif and Ammar Muthanna

Received: 5 February 2023

Revised: 28 February 2023

Accepted: 6 March 2023

Published: 14 March 2023



Copyright: © 2023 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

Software-Defined Networking (SDN) is a new network architecture consisting of three separate planes, the application plane (AP), control plane (CP), and data plane (DP). The CP is in the middle of the architecture and acts as the network's operating system (NOS) responsible for managing the network devices at the DP based on the network policies hosted at the AP. These policies could be for managing various aspects of the network operations such as routing, QoS provisioning, load balancing, and security. The communication between AP and CP is performed through a northbound interface (NBi) while that of the CP and DP is performed through a southbound interface (Sbi). The DP is relieved from all control functions and focuses only on forwarding traffic from

source to destination based on the CP's instructions. For that, the CP acquires the network statistics in real time to formulate the required operational policies and install them as rules in the flow tables of the devices at the DP. Some of the earliest SBI used in SDN were Protocol Oblivious Forwarding (POF) [1], Open vSwitch Database (OVSDb) [2], Forwarding Control Elements (ForCES), and OpenFlow [3]. Many SDN controllers use OpenFlow as SBI, through a Link Layer Discovery Protocol (LLDP), to discover and build a topology of the DP switches. The controller maintains a consistent global knowledge of the topology discovered at all times [4]. Upon the arrival of any flow, it activates a Path Selection Algorithm (PSA) to compute routing instructions for the flow and install it in the switch's flow table. The PSA is always invoked whenever a new flow with no corresponding entry in the flow table arrives. Another reason could arise from topology changes due to link or node failure. In both situations, the PSA is required to converge the network with the new rule to avoid disruptions. The controller keeps tabs on these changes with the help of a thread monitoring mechanism, which collects network statistics in a constant cycle. The monitoring mechanism periodically sends a request to the switches. It feeds the acquired information into the controller for the new rule computation.

Isyaku et al. report in [1] that network rules need to be updated often, in fact, every 1.5 to 5 s on average. The short interval update rate is due to increased traffic volume and demand for ubiquitous services in contemporary networks. This increase and demand are influenced by reliance on digitalized lifestyles as the proliferation of the Internet of Things (IoT) [5] and smart environment applications [6] are on the rise. In addition to the traffic volume, the traffic flow is heterogeneous as it displays asymmetric patterns regarding arrival rate, duration, and size. As a result of this heterogeneity, the threats and vulnerabilities [6], requirements for path quality, and switch flow table space of flows, among other network resources, are different. In other words, they possess distinctive QoS requirements and behave differently en route to their destination. For example, large flows, such as Elephant Flows (EF), demand an exorbitant amount of network resources despite accounting for only 1 to 10% of overall traffic. These flows are long-lived (LLF) and have the propensity to consume network buffers rapidly, thereby causing anomalies such as congestion and queuing delays for most small-sized Mice Flows (MF) [7]. Similarly, despite being short-lived, the Mice Flows form the majority. They are delay-sensitive, requiring prompt delivery to their destination. Due to these disparities in behavior, heterogeneous traffic flows must be treated differently so that each Service-Level Agreement (SLA) may be satisfied. That is, time-sensitive flows must get to their destination without any significant delays. In contrast, high-volume traffic flows are directed down routes with adequate capacity to avoid congestion in the network.

The existing route path selection methods often use a single metric, such as the shortest path based on hop count [8], latency [9,10], or bandwidth [11], to make decisions about which paths to take. Other approaches combined the link feature with the switch update operation to guide the path selection decision to reduce convergence time [12–16]. However, these methods do not differentiate flows according to their unique attributes. The differentiation is necessary to avoid the congestion that might arise by amalgamating EF with MF on the same path. The classification is also necessary for ensuring differential QoS provisioning to the heterogeneous traffic flow. Meanwhile, the techniques such as [17–19] that attempt to classify the traffic flow for differential handling consider 10% of link utilization regarding a single flow feature such as size as a baseline threshold for determining whether a flow is congestion susceptible or not. However, considering a single flow feature and a fixed threshold value for flow classification might lead to premature, fake, or unnecessary detections that might cause additional overhead on the controller. Therefore, the detection approach, in conjunction with the path selection metrics used, is not an appropriate solution for all types of traffic flows with varying degrees of criticality. Some flows require high throughput, while others are delay sensitive. Failure to classify these traffic flows and treat them differently might lead to QoS violations.

In Software-Defined Networking, the adaptive path selection Algorithm with flow classification is a crucial mechanism for improving network performance and reliability. This Algorithm dynamically selects the best path for data transmission based on real-time network conditions and flow characteristics, such as traffic volume, delay, and bandwidth. By combining flow classification with composite path selection metrics, SDN can achieve improved resource utilization, reduce congestion, and enhance the Packet Delivery Ratio (PDR) and throughput of the network. Thus, this paper proposes an Adaptive Path Selection Algorithm with Flow Classification (APSAF) by incorporating flow classification features and composite path metrics (comprising delay, delivery ratio, and residual capacity) to address the outline issues. The contributions of this paper can be summarized as follows.

- A traffic flow classification module to separate Elephant Flows from Mice Flows.
- Designed a composite path quality estimation vector to evaluate path suitability to accommodate traffic class.
- The paper designed a differential path selection scheme based on traffic flow propensity to congestion.

The remainder of the paper is structured as follows: Section 2 discusses current path selection strategies in SDN. Section 3 describes the proposed solution's design. Section 4 describes the experimental setup and performance evaluation. Lastly, Section 5 concludes the study and makes recommendations for future research.

2. Related Works

Path Selection Decision (PSD) to route traffic is crucial to the operation of a network. In SDN, the decision is taken by a controller on the arrival of any flow with no corresponding entries in the switch's flow table. Traffic volume and heterogeneity necessitate this activity to occur regularly [1]. The heterogeneities in terms of burst, rate, size, and duration exhibited by flows make their behavior and demand on network resources en route to their destination different [20]. Failure to consider these variabilities may lead to the choice of a non-optimum path that can ultimately violate the requirement of the traffic flows. For instance, 95% of network traffic emanates from the 1-10% large size (EF) traffic [21]. Although very few, EF traffic flows cause serious congestion problems in a network as they tend to live long (LL) on their paths en route to their destination [22]. For that reason, these flows ought to be detected and differentiated from others (MF) and rerouted separately on an alternate path. Schemes such as [23–27] and [14] overlooked these requirements while performing flow scheduling. However, detecting these flows and mapping them to the proper path is challenging. There are three approaches to detecting these flows. Detection at the Edge Switch (ESD) via polling-based statistic monitoring, detection at the End-Host (EHD) via host-based monitoring, and Switch Trigger Detection (STD). Past literature shows several studies have proposed different PSD techniques based on specific routing parameters.

Famous techniques such as Equal Cost Multiple Path (ECMP) [15] distribute traffic load among multiple equal cost paths in a network. However, ECMP does not regard the variabilities of flows while taking the PSD. The mechanism amalgamates multiple flows, irrespective of their requirements and behavior, on the same path en route to destinations. This action led to switch buffer overflows, inefficient use of network bandwidth, and eventual degradation of network performance. To address these issues, Refs. [26,27] used the STD approach to classify flows in their PSA. However, STDs require a specialized cloned switch to set up a detection threshold in advance. Furthermore, due to the dynamic nature of the network environment, it is challenging to formulate a suitable threshold value that can give precise and accurate detection output. In addition, continuous cloning of the switch flow table can burden them more. In contrast, the flow rerouting scheme proposed by [13] used ESD. However, employing ESD makes the proposed solution suffer from late detection of congestion-prone flows. The detection is not attempted until the flows arrive at an edge switch. Thus, collision is still experienced. Furthermore, a monitoring overhead because of per-flow statistics collection is also experienced in addition to the long latency and

packet drop. Conversely, the alternate approaches by [28,29] used EHD. EHD introduces an additional layer called SHIM to achieve early detection at the end host. However, the approach requires a specialized and expensive hardware mechanism modification. As such, such approaches experienced low adaptation to flow behavior, scalability, and a complex host operating system modification process. To avoid the dilemma of choosing between EHD or ESD approaches. Refs. [30,31] proposed an alternative approach using traffic split. Instead, to detect the flow rate variation, the authors proposed a scheme that splits the traffic into multiple flow streams and places them simultaneously along multiple paths. In contrast, these schemes might record fine-grained flow control of congestion but at the expense of additional overhead of flow reassembling at the destination. Similarly, it might not work for the adaptation window of transmission control protocol (TCP) flows, especially when packets of the same flows embark on distinctive paths.

Many approaches such as RPSO [14], DASH [20], MINNIE [32,33], [22] and Banerjee and Kannan [34,35], consider the limitation of Ternary Content Addressable Memory (TCAM) in terms of capacity while designing their PSA. DASH is a hashing-based implementation of a traffic split designed to prevent memory overflow by matching the hash to range bounds. The technique is an adaptive weighted traffic partitioning Algorithm that operates in the DP. MINNIE is another rule space reduction approach to select load-balanced routes. The technique comprises a two-phase solution made up of a compression phase and a routing phase. Other complementary solutions to MINNIE are proposed by Banerjee and Kannan [34,35]. Using a Path Tag (PT) and a Flow Tag (FT), the authors devised a tag-in-tag technique to reduce the number of bits representing a flow within a switch, hence reducing the size of flow rules (FT). The PT is used for routing the packets while FT is used to map each packet with its corresponding flow path. However, adding an identifier to each incoming packet is hard to achieve in the ASICs since this is not a standard operation, causing the packets to be processed by the CPU (a.k.a. The slow path), strongly penalizing the performance of a forwarding device and the traffic rate.

In a multipath approach, a Dynamic Multipath Scheduling Protocol (DMSP) for identifying and isolating congestion-susceptible links in DCN using SDN is demonstrated in [25]. DMSP splits flow traffic among multiple paths to reduce congestion. However, the split traffic process is conducted unequally among the available paths. To address the unequal split problem, Ref. [26] proposed a Globally Optimized Multipath Routing (GOMR) Algorithm that splits a flow traffic equally among multiple paths using a stochastic mechanism. GOMR leverages the global knowledge of network topology and traffic statistics to formulate the problem as linear programming (LP). However, though the technique distributes the load along multiple paths, it did not classify the traffic according to their requirements and the burden they introduced to the links. Recently, Kamal and Taha [27] proposed an adaptive method that computes multipath routes for video traffic with a reduction time objective while recovering from link failure in SDN. The method, however, did not consider congestion risks associated with Elephant Flows. HiQoS [36], ADMPCF [37], TALON [38], and Presto [28] are other PSA for multipath selection using a Traffic Engineering (TE) tool to guide differential treatment of flows. HiQoS compute the multipath (at least two constrained pathways) between all pairs in a network, allowing rapid recovery from link loss. The adaptable and Dynamic Multi-path Computation Framework (ADMPCF) analyses and extracts information from traffic flows to enhance resource utilization, alleviate congestion, and guarantee QoS when network events such as link failure occur. On the hand, TALON is a traffic LB approach with split multipaths for high throughput. TALON collects the trajectory information and assigns a flow capacity per tenant by calculating multiple routes to satisfy the traffic requirement. However, although TALON enhanced capacity by allocating tenant throughput, it ignored latency and reliability. While Presto leverages ECMP to optimally balance end-host load in asymmetric topologies by partitioning each flow into equal flowcells at the edge and equally distributing them to the network, Presto employs the maximum TCP Segment Offload (TSO) size (64 KB) as the default flowcell size, allowing fine-grained load balancing at

10+ Gbps. The mechanism merges out-of-order packets into segments and pushes them up at the receiver using the GRO algorithm. However, transforming the flows into equal-size mice might not be adequate despite assigning the flowcells evenly to spanning trees from each edge switch, because flows coming from discrete soft edge switches are not uniform, thus, preventing the reordering at the receiver to detect out-of-order packets and missed packets, which will lead to longer flow latencies. In addition, Presto is suboptimal to handle asymmetric topologies even if its weighted Algorithm can drive some loads away from congestion paths. Recently, Yoo et al. [29] presented TeaVisor, the first bandwidth isolation guarantee framework for SDN virtualization with three components: path virtualization, bandwidth reservation, and path establishment.

On the other hand, Ref. [30] designed a mechanism that finds a controller's flow setup delay. The authors crafted a timestamped Special Ping Packets (SPP) from the controller. The Round-Trip Time (RTT) of SPP back to the controller is measured and attached to an individual path to get the total path setup delay. The technique is designed for scCP. In another work, Ref. [31] introduced a greedy heuristic technique based on Yen's k -shortest path algorithm. Multiple metrics such as packet loss, delay, and bandwidth are jointly considered to formulate a QoS-aware routing problem as Integer Linear Programming (ILP). They validate the technique using POX [39] on Mininet with D-ITG [32] as an IoT-based traffic generator. In a similar effort, Ref. [33] study the effects of various optimization objectives on network performance. Performance metrics that include link usage and latency are considered constraints in the study. Another constraint considered is the switch related in terms of TCAM size. The constraint is modeled so that the number of outgoing flows is less than the maximum number of forwarding table entries. Then, a path selection technique bounded by these constraints is designed. However, in large networks with frequent topology changes, the use of the ILP-based method is impractical because it may slow the convergence of the network routing rules [34]. Ref. [35] improved on [30] to address CP overhead, scalability, and Single Point Failure (SPOF) by considering distributed CP with multiple controllers. However, both schemes may experience higher flow table operation time, affecting path setup switching time. In [40], packet loss and latency were considered to design a Path Selection Method (PSM) in an SDN-Edge computing integrated environment. The edge node at the SDN boundary is configured to assign available network resources, such as bandwidth, according to flow requirements. Flows are re-directed via a path with lower packet loss, while delay-sensitive flows are routed with minimal delay. Likewise, Ref. [41] proposed an incremental QoS-aware path selection technique to facilitate a speedy redirection of real-time applications with time-bound flows. The technique avoids bottleneck links that were the cause of the scheduling impasse by selecting a path with enough residual bandwidth from the list of candidates' paths. The technique involves an offline pre-routing stage where initial K -paths are formed using Yen's K -SP [42]. The run time of this stage is observed to be high. However, the authors attempted to control it by coupling a Fibonacci Heap with Dijkstra [43] and the [42]. However, EF was not separated from MF when choosing a path in all these techniques. Flows must be classified accordingly for a better link choice. Likewise, QoS parameters such as throughput and the link delivery ratio should also be considered. In a similar technique [43], the data transmission (Tx) and receive rates (Rx) of a port are observed to help in determining the maximum traffic load the port can accommodate. Flow statistics are collected and fed to a Floodlight controller application [44] to define rules that can reroute the flow via a port with Least Loaded Path (LLP). In validation, the authors used Mininet to emulate OvS and Iperf to generate traffic. On the other hand, a QoS-driven and SDN-assisted Multipath Selection Scheme (QSMPS) was proposed [45]. QSMPS addresses the adaptability problem of traditional MPTCP. The method constantly checks network status using a scalable SDN-assisted technique. Based on the data, an optimal number of sub-flows is determined and distributed along routes with negligible differential delays. The authors validate the technique in a custom topology in Mininet with the Ryu controller framework. However, a regular gathering of statistics may significantly increase the controller overhead, which will increase flow

setup latency. Similarly, a best-case scenario based on residual bandwidth might not always ensure that distinctive flows' requirements are met, because the scheme did not primarily classify traffic according to its uniqueness. Other flows can choose a setup latency that is as short as possible. In addition, high link quality decreases the demand for link change requests in a network.

Some techniques [46] adopt a fixed load threshold in the detection process of these flows. The threshold is defined regarding a link bandwidth. Initially, all incoming flows are routed based on traditional protocols such as ECMP [15]. The ECMP is maintained until flows increase and grow beyond the predefined threshold. Then, the controller re-computes an alternate path to reroute the detected flows. However, employing a fixed threshold value to detect the flows susceptible to congestion is inefficient in highly dynamic traffic conditions. Depending on the threshold size, the fixed value might not be adaptive to flow variability and heterogeneity of controllers' capacities in terms of their flow processing ability at a particular time. The inflexibility of the fixed threshold value might not correctly reflect or predict congestion occurrence possibility at a particular time. Therefore, it may prompt early detection or delayed detection of flows to startup a rerouting action. Controller overload might occur when the threshold is too low because many flows will be considered EF. In such circumstances, path re-computation and flow rerouting instance frequency will increase. On the other hand, if the fixed threshold value is set too high, many high-risk EF might evade detection and bring about undesirable congestion.

It can be noticed from Table 1 that none of the available research has considered combining the path quality metrics and flow classification technique in the path selection decision. Path quality is severely impacted in a dynamically evolving network with heterogeneous traffic, resulting in congestion. Consequently, it is crucial to consider path quality and flow categorization while selecting routing paths. It is a non-trivial challenge since path quality and flow classification provide information regarding the network's ability to accommodate increased demand as it evolves. This research proposes an optimum route path selection strategy based on composite path quality parameters and classification of flows according to their tendency to cause congestion.

Table 1. Summary of Related Works.

Existing Approaches Algorithm	Objective	Methods Classification				Routing Algorithm/Approach	Result Achieved	Weakness
		Traffic Flow		Path Selection				
		Classification	Split	Link Quality	Switch Feature			
Reliable Path Finder (RPF) [8]	To reduce backup path installation time, number of rules, and controller’s overhead during failover	N	N	N	Y	Context-Aware	NA	Ignore flow classification
Q-Learning [47]	Optimum path selection	N	N	N	Y	Q-Learning	Improved bandwidth by 38.10%, latency by 19.10%, and loss by 10.81%	High memory demand
Sorted-GFF [46]	Reduce congestion	Y	N	Y	N	Global First Fit	Improve bisection bandwidth	Inefficient flow classification & prioritization
Flow Setup Latency (FSL) [30]	To reduce flow setup latency	N	N	Y	N	Context-Aware	Reduce median and percentile latencies to 5.9 and 7 ms, respectively	High path convergence time
RPA [48]	To improve link usage and traffic distribution	Y	N	Y	N	Heuristics	Claimed to have improved delay and packet loss compared to ECMMMP and GFF.	No empirical evidence to support the claims.
PDMR [49]	To Improve network load, jitter, and packet loss	Y	N	N	N	Heuristics	Improved network load by 0.075 and 0.173, jitter by 0.191 and 0.407, and packet loss by 0.020 and 0.049 for QoS and best effort flow, respectively.	Failure to treat EF separately.
VPSA [10]	To improve video QoS experience	Y	N	N	N	Holding Times and SDP	Claimed to have improved bandwidth	High path set up time
Path Switching Time [50]	minimizing total path switching time	N	N	N	Y	Context-Aware	NA	Poor validation
TDR [51]	optimize aggregate delay and reliability	Y	N	N	Y	Ant Colony	Not Clear	Only suitable for WSN
MCRA [52]	minimize congestion and wasted bandwidth	Y	N	Y	N	Bellman Ford Algorithm	Improve throughput, delay,	Single Fixed Flows Feature during classification, not scalable
RPSO [14]	Optimize overhead and convergence time	N	N	Y	Y	Context-Aware	Reduce path stretch by 37%, Latency by 73%, throughput by 55.73%, and PDR by 12.5%	Overlooked flow features and this affect the QoS of important flows
MRA [27]	To Reduce failover time	N	N	Y	Y	Depth First Search	Achieved failure recovery within 1ms	Ignored congestion-prone flows
HiQoS [36]	For Rapid recovery from link loss.	Y	N	Y	N	Dijkstra	Reduce delay, and Increase throughput	Did not consider congestion-prone traffic such as EF

Table 1. Cont.

Existing Approaches Algorithm	Objective	Methods Classification				Routing Algorithm/Approach	Result Achieved	Weakness
		Traffic Flow		Path Selection				
		Classification	Split	Link Quality	Switch Feature			
ADMPCF [37]	Integrated resource control	N	N	Y	N	BGR, kSP	Outperformed H-SPF and W-SPF on rejected number of requests by 81–83%, SPF and H-SPF in utilization by 1.4% and 40%, respectively.	BGR needs additional memory to store auxiliary graph
TALON [38]	To improve throughput among tenants	N	Y	Y	N	ECMP	Increased throughput by 2.29	Did not consider latency and reliability
Presto [28]	Load balancing	N	Y	N	N	GRO	For shuffle synthetic workload, Presto improved throughput upon ECMP by 38–72% and MPTCP by 17.28%	Is a proactive approach and suffers from flowcell reordering bottleneck. Thus, it is prone to memory overflow
TeaVisor [29]	Bandwidth reservation	N	N	Y	N	Context-Aware	Achieve 34% throughput improvementAverage error rate 5%	
DASH [20]	Flow table load balancing across multiple paths	N	Y	N	Y	Hash-based data structure	Achieved 22.1% and 16.0% completion time at 80% workload compared to Hula and ECMP. With a 20% load balance	Prone to out-of-order packet delivery.
Tag-in-Tag [34,35]	To maximize flow table	Y	N	N	Y	Context-Aware	accommodates 15× more flow entries and reduces power consumption by 80%	Adding an identifier to each incoming packet is hard to do in the ASICs
Minnie [21]	Efficient memory utilization	N	N	Y	Y	Wildcard	Save 50% of memory, decrease the number of rules by an order of 5, and 31 for dynamic compression at 2000	

3. Design of the Proposed Solution

The proposed Adaptive Path Selection Algorithm with Flow Classification (APSAF) is integrated into the SDN controller for routing traffic. APSAF employs a lightweight network monitoring utility in the controller. The tool gathers and analyses network statistics in real time to guide the selection of a suitable path for routing traffic while being aware of its demand and tendency to cause congestion. Based on the statistics, traffic flows are classified into two (2) categories by considering multiple features (size, rate, duration, and priority) to distinguish the flows prone to congestion from the others dynamically. Furthermore, APSAF incorporates multi-facet path features suitability metrics (PSM) to enable path screening and selection based on the requirements matching each flow's demands. The working mechanisms of APSAF involve a Data Plane Elements Discovery Phase (DPEDP), a Flow Classification Phase (FCP), and a Path Screening and Selection Phase (PSSP). The output of each phase served as an input to the next phase. The diagram in Figure 1 displays the architectural view of the APSAF in the SDN framework. Similarly, the description of the proposed Algorithm is provided in the flowchart shown in Figure 2. Hence, each phase description and detailed operational procedure are provided in Sections 3.1–3.3.

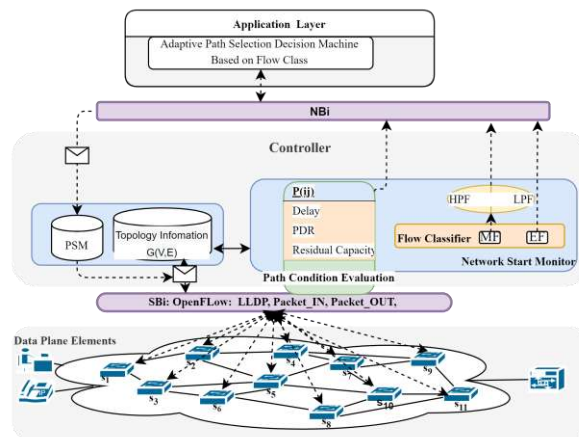


Figure 1. Architecture of APSAF.

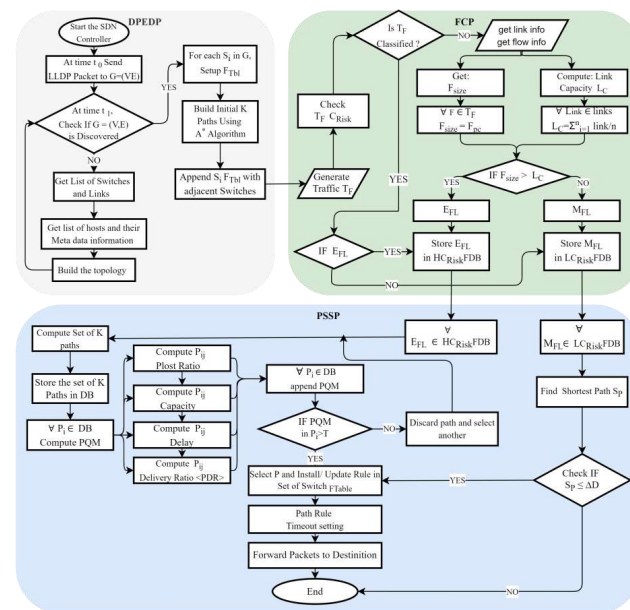


Figure 2. Flow Chart of the Proposed Algorithm.

3.1. Data Plane Elements Discovery Phase (DPEDP)

The Data Plane (DP) elements represent the network topology, which consists of a collection of hosts $H = \{h_1, h_2, h_3, \dots, h_{|H|}\}$ and switches $V = \{s_1, s_2, s_3, \dots, s_{|V|}\}$. Their topology is modeled as a graph $G = (V \cup H, E)$ where E is a set of links $E = \{e_1, e_2, e_3, \dots, e_{|E|}\}$ connecting any two switches s_{ij} . Each of these links $e_{i,j} \in E$ has a capacity $C(e_{i,j})$ and is responsible for carrying traffic flow T_f from a source switch s_i to destination switch s_j . These elements are discovered and built into a topology using the procedure described in DPED of Figure 2 and Algorithm 1. When topology discovery status $TDt_x = 0$, the SDN controller initiates a Link-Layer Discovery Protocol (LLDP) to discover the network elements' presence at the DP. Beginning with the initial handshake, the controller broadcasts a Feature_Request_Message, to which all the available switches respond with Feature_Reply_Message. The replied messages contain information such as (Switch-ID, Active-Ports, MAC-Address, and Port-Number) to aid subsequent discovery of links. With this knowledge, the controller encapsulates and multicasts an LLDP packet to all the Active-Port of all the switches that responded to the Feature_Request_Message using Packet-Out message *Packet_OutMsg* (Line 1–3, Algorithm 1). Suppose S is the total number of switches that responded to the request in the network. In that case, the network, the total number of LLDP *Packet_OutMsg* multicast to all the Active-Ports of the S number of switches discovered, is given by Equation (1):

$$Packet_OutMsg_{Total} = \sum_{i=1}^S Active_Port_i \tag{1}$$

Additionally, the *Packet_OutMsg* is also used to install an initial flow rule entry in the switches' flow table to route the *LLDP_Packet* via a *Port_ID* of any adjacent switches as indicated in Type Length Value *TLV_field* of the *LLDP_Packet*. Similarly, any switch upon receiving the *LLDP_Packet* via a *Port_ID* not belonging to the controller will compose a *Packet_INMsg* to the controller. Hence, the total number of *Packet_INMsg* is twice the number of links L as shown in Equation (2).

$$Packet_INMsg_{Total} = 2.L \tag{2}$$

The message contains the switch meta-data info such as *Switch_ID*, *Port_ID*, *Link_info*. This information is updated after pre-set discovery interval $TD_{Interval}$. (Line 3 Algorithm 1). Figure 3 illustrates the signaling for these message exchanges.

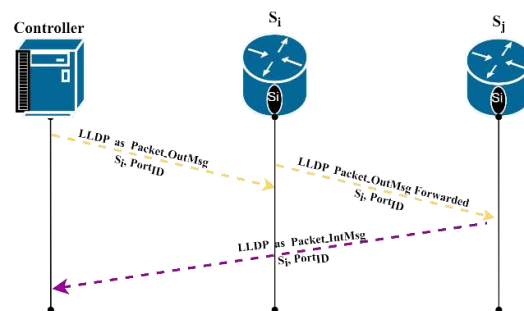


Figure 3. LLDP Signaling During Topology Discovery.

Therefore, through this process, the controller at a time $TDt_x = 1$ gets all the information required to build the global network topology $G = (V \cup H, E)$. The information comprised the list of all adjacent switches $s_i \in V \forall i = 1, 2, \dots, |V|$, links $e \in E$, and their meta-data to store in its local data structure as implemented by (Lines 8–10 Algorithm 1). Similarly, for each switch $s_i \in V$, a flow table is maintained that contains a list of all

its adjacent switches, and all K paths P from s_i to s_j , built with the A^* [53] as shown in (Lines 14–19, Algorithm 1).

Algorithm 1 Data Plane Element Discovery (DPED)

1. **Begin**
 2. Set Discovery Status time $TDt_x = \begin{cases} 1 & \text{if } G = VE \text{ discovered} \\ 0 & \text{if } G = VE \text{ not discored} \end{cases}$
 3. Set Discovery interval $D_{Interval}$
 4. Initiate LLDP to Discover $G = VE$
 5. // Get Topology
 6. **While** (*True*):
 7. Sleep for $TD_{Interval}$
 8. Get switches $s \in V$
 9. Get links $e \in E$
 10. Get Hosts
 11. **End While**
 12. //Build the Topology
 13. **IF** $TDt_x = 1$ **Then**
 14. **For** $s \in V$ and $e \in E$
 15. Get Adjacent switches.
 16. Prepare a $Flow_{table}$
 17. Build Initial paths P_{ij} using A^* algorithm.
 18. Install P_{ij} in $Flow_{table}$
 19. **End For**
 20. **Else**
 21. Repeat 6 to 11
 22. **End IF**
 23. **End**
-

3.2. Flow Classification Phase (FCP)

The Flow Classification Phase's (FCP) responsibility is to classify traffic flows according to their characteristics and tendency to cause congestion. Although, in electronic communication, a packet is the basic unit of data encapsulating the messages sent from a source to a destination, because SDN is a flow-based networking framework, the network traffic is treated in flow teams. Therefore, in this study, a flow is considered a sequence of packets carrying information from source to destination whose packets share the same 5-tuple. These tuples cover its Protocol (TCP, UDP, ICMP), Source-Destination IP, and port numbers subjected to the same timeout in a switch flow table. Flow characteristics in SDN are not uniform regarding arrival rate, size, path use duration, and demands for resources such as bandwidth. Owing to these irregularities, they posed different congestion threats, possessed different QoS requirements, and behaved differently en route to their destinations. The rule for handling each of these flows is stored in the switches' flow table using the following format ($dpid(match), priority, instruction, counters, timeout cookie$). The $dpid$ identifies the switch, while the $cookie$ is a distinctive value to identify each flow. Other fields recorded at the time of entry are priority and timeouts. Additionally, the flow table provides counters to track the number of packets and bytes handled by each port and rule. The statistics are encoded using OpenFlow Extensible Stat (OXS), a compact type-length-value (TLV) format.

Therefore, the FCP Phase, as described in Algorithm 2, acquires and inspects the characteristics of each traffic flow Tf_i coming into the network as stored in the flow table and classifies them into either high congestion risk HC_{risk} or low congestion risk LC_{risk} .

The FCP procedure begins upon arrival of any new traffic flow Tf_i , through any of the ingress switches of the network $G = (V, E)$. Two containers are created to hold

only the information related to active traffic flows (Lines 2–3, Algorithm 2). Suppose $Tf = \{Tf_1, Tf_2 \dots Tf_N\}$ are the Set of N traffic flows coming into the network. Each $Tf_i \in Tf$ is classified either as a high congestion risk HC_{risk} or a low congestion risk LC_{risk} , based on its size, arrival rate, and path utilization duration. The Algorithm checks whether the traffic flow is classified in (Lines 8–10, Algorithm 2). Inspired by [54,55], the packet counts ($Pcount$) of each Tf_i are used to quantify the size of the traffic flow Tf_{size} .

To get the traffic flow information for this classification, a Ryu controller inbuilt utility that gathers individual and aggregate flow statistics from switches is used. Therefore, to bring out the flow of information, $\forall Tf_i \in \{Tf_1, Tf_2 \dots Tf_N\}$, an $OFPStatsRequest$ is sent in synchronous mode at a different time interval. Moreover, a reply is received through $OFPStatsReply$ messages for each request made. Accordingly, the traffic flow size Tf_{size} and duration $Tf_{duration}$ are extracted from the data obtained (Line 12–13, Algorithm 2) while the traffic flow arrival rate, $Tf_{ArrivalRate}$, is determined by dividing the flow size Tf_{size} of the transmitted data by the flow duration $Tf_{duration}$. If $Tf_{ArrivalRate} > Th$, the flow is a cheetah and snail otherwise.

$$Tf_{ArrivalRate} = Tf_{size} / Tf_{duration} \tag{3}$$

A flow is classified as a high congestion risk if its size and duration are more than fifteen (15 pkts) packets and eleven (11 ms) milliseconds, respectively. Otherwise, it is classified as a low congestion risk flow. The procedure is shown in (Lines 16–20, Algorithm 2).

Algorithm 2 Flow Classification Phase (FCP)

```

1. Begin
Input: a set of all new flows  $Tf \{Tf_1, Tf_2 \dots Tf_N\}$ 
Output: Flow Categories  $HC_{risk} [ ], LC_{risk} [ ]$ 
2. Set  $HC_{risk} \leftarrow$  High congestion risk flow.
3. Set  $LHC_{risk} \leftarrow$  Low congestion risk flow.
4. IF new Traffic Flow  $Tf$  arrived Then
5.     Call Algorithm 1
6.     Get a set of  $K$  path  $P_{ij}$ 
7. End IF
8. For each new  $Tf$ 
9.     IF  $Tf$ , is classified, Then.
10.        Find a suitable path (Call Algorithm 3)
11.     Else
12.        Send_FlowStat request.
13.        Extract the  $Tf_{size}$ 
14.        Extract the  $Tf_{duration}$ 
15.        Compute link utilization value  $L_{uv}$ 
16.        IF  $Tf_{size} > 15$  and  $Tf_{duration} > 11$  Then
17.             $HC_{risk} \leftarrow Tf$ 
18.        Else
19.             $LC_{risk} \leftarrow Tf$ 
20.        End If
21.        For each  $Tf \in HC_{risk}$  or  $\in LC_{risk}$ 
22.            Find a suitable path (Call Algorithm 3)
23.        End For
24.        For each  $Tf \in LC_{risk}$ 
25.            Find a suitable path (Call Algorithm 3)
26.        End For
27.    End IF
28. End For
29. End

```

3.3. Flow Adaptive Path Selection with Optimized Quality

The Adaptive Path Selection Algorithm with Flow Classification and Optimized Quality's (FAPSOQ's) responsibility is to route the flow traffic from the source s_i to destination s_j upon arrival of new flow in the network $G = (V \cup H, E)$. At the arrival of any new flow, Algorithm 1 will be called upon to return the Set of K candidates $paths P\{p_{ij}, p_{ij} \dots n\}$ and

Algorithm 2 to classify the flows according to their congestion risk tendency. Algorithms 2 will return the flow classification result as HC_{risk} and LC_{risk} , respectively. The procedure is shown in (Lines 1–7, Algorithm 3).

To route, the flow of traffic classified as high congestion risk and stored in HC_{risk} . All the set of candidate paths returned by Algorithms 1 is further screened for the quality-of-service requirements of congestion-susceptible flows; i.e., for each $p_{ij} \in P\{p_{ij}, p_{ij} \dots n\}$, a composite QoS metric called Path Suitability Metric (PSM) is estimated and appended to the path accordingly. PSM is a path feature vector defined by the path Minimum Delay ($\min p_{ij} D$), Maximum Delivery Ratio ($\max p_{ij} DR$) and Maximum Residual Capacity ($\max p_{ij} C$) as shown in (Lines 9–15, Algorithm 3). FAPSOQ selects the path with maximum PSM and installs the corresponding rules in the switches' flow table along that path using a command $F_{install}$ to begin routing flows to their destinations (Line 16–17, Algorithm 3) while for flows classified as low congestion risk and stored in the LC_{risk} container, they are routed using the Dijkstra [56] as shown in (Lines 20–23, Algorithm 3). The procedures used to calculate path Delay ($\min p_{ij} D$), Delivery Ratio ($\max p_{ij} DR$), and Residual Capacity ($\max p_{ij} C$) are as follows:

To calculate the $p_{ij} D$ and $p_{ij} DR$, an Open Link Layer Discovery Protocol LLDP [57] looping method for latency monitoring is used. A timestamped LLDP packet with an extra Network Specific Type Length Value (TLV) is crafted and sent as a continuous probe message to measure all links' latency. Adding the extra TLV in the normal LLDP enables specifying a 10 bytes Organizational Unique Identifier (UOI) and Defined Subtype (ODS) TLV to record the timestamp of LLDP packets traversing any link for latency measurement.

The DP switches are connected to the CP in an Out-Band mode. The controller is used to inject a timestamped LLDP Probe Packet (PP) with $TTL = 4$ to switch S_i designated as the Link Source Switch LSS_i at the DP. On receiving the PP from the controller, switch LSS_i forwards it further downstream to all its adjacent switches $S_j, S_k, \text{ and } S_l$ called LDS_j via all links connected to its active ports. Depending on the TTL value, a switch can either be a Link Source Switch LSS or Link Destination Switch LDS. For a switch to know its role/status, the controller crafted the LLDP PP with $TTL = 4$ at the beginning.

A PP arriving at any switch with $TTL = 4$ implies that the LLDP message emanates from the CP and the switch is an LSS. The switch will decrease the TTL by 1, flood it to its active ports, and forward it downstream to LDS. Furthermore, the TTL value controls how the PP loops around a link to enable the calculation of its latency at the LDS; see the flowchart in Figure 4.

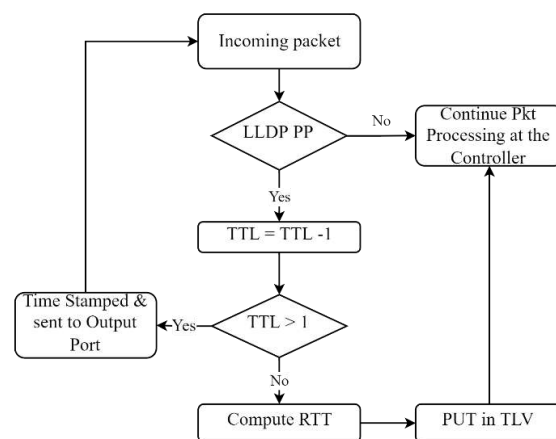


Figure 4. LLDP Looping Flowchart.

A switch S_i is tagged as an LSS, and a switch S_j that received the LDDP probe packet from S_i is tagged as an LDS. At S_j , the probe packet arriving with $TTL = 3$ signifies that it is an LDS receiving the packet and is required to record the time it received the packet, decrease the TTL by one (1) and return the packet to the LSS_i . A probe packet arriving at

any switch with TTL = 2 implies that the LLDP message emanates from the LDS_i , and the switch is an LSS_i , receiving the packet for the second time. The switch will decrement the TTL by 1, flood it to its active ports, and forward it downstream to LSD . On seeing the PP with TTL = 1, the LDS_i LDS will compute the RTT, put it in the TLV, and return it to the controller. The time it takes for the controller to receive the probe packet from LDS, T_{LDS-C} , is measured. Therefore, the total delay of a link between the LSS and LDS along any path is accumulated by the time difference between PP sent time at LSS, and PP received time at LDS; $LSS_i ST - LDS_j RT$ and T_{LDS-C} , as shown in Equation(4). Hence, for a path with multiple links, the total path delay $p_{si-sj} D$ is obtained by taking the summation of all links' delay along the path, as shown in Equation (5).

$$L_D = T_{LDS-C} - \frac{(LDS_j RT - LSS_i ST)}{2} \tag{4}$$

$$p_{si-sj} D = \sum_i^j L_D \tag{5}$$

Similarly, as shown in Equation (6), the $l_{si-sj} DR$ is computed by taking the ratio of the total number of LLDP Probe Packets received at s_j (*No. of PP Recieved s_j*) sent over the link l_{si-sj} from the sender s_i (*No. of PP Sent s_i*) at a particular time. The statistics are collected by sending an OpenFlow *Port Starts Request* message specifying the port numbers of the LSS and LDS.

$$l_{si-sj} DR = \frac{\text{No. of PP Recieved } s_j}{\text{No. of PP Sent}_{s_i}} \tag{6}$$

$$p_{si-sj} DR = \sum_i^j l_{si-sj} DR \tag{7}$$

While the Path Utilization Ratio $p_{si-sj} U$ is determined by using an OpenFlow [58] *Port Starts Request* feature to send a request to a LSS at time interval t , the port number connected to the link of interest is specified in the request to retrieve the number of bytes transmitted. If ρ is the number of bytes transmitted by LSS over the link l_{si-sj} to LSS and δ is the period between the two times at which ρ is polled from LSS, then the utilization of the link $l_{si-sj} U$ and $p_{si-sj} U$ is calculated as shown in Equations (8) and (9), respectively.

$$l_{si-sj} U = \frac{\rho[t] - \rho[t-1]}{\delta} \tag{8}$$

$$p_{si-sj} U = \sum_i^j l_{si-sj} U \tag{9}$$

If the capacity of link $si \rightarrow sj$ is C , the Link Residual Capacity and Path Residual Capacity are calculated by Equations (10) and (11), respectively.

$$p_{si-sj} rC = C_{si-sj} - l_{si-sj} U \tag{10}$$

$$p_{si-sj} rC = \sum_i^j p_{si-sj} rC \tag{11}$$

Therefore, aggregation of Equations (5), (7) and (11) leads to the formation of composite Path Suitability Metrics (*PSM*) as shown in Equation (12)

$$PSM = \sum_i^j ((\min(p_{si-sj} D)) + (\min(p_{si-sj} DR)) + (\max(p_{si-sj} rC))) \tag{12}$$

Algorithm 3 Adaptive Path Selection Algorithm with Flow Classification

1. **IF** new flow T_f arrived **Then**
2. Call Algorithm 1
3. Get Set of K paths $P\{p_{ij}, p_{ij} \dots n\}$
4. Call Algorithm 2 to Input
5. Get HC_{risk}
6. Get LC_{risk}
7. **End IF**
8. **IF** $T_f \in HC_{risk}$ **Then**
9. **For Each** $p_{ij} \in P$
10. // Compute the values of $p_{si \rightarrow sj} D$, $p_{si \rightarrow sj} DR$, $p_{si \rightarrow sj} rC$ and $p_{si \rightarrow sj}$ lost for $PSM_{si \rightarrow sj}$
11. $p_{si \rightarrow sj} D = \sum_i^j T_{LDS-C} - \frac{(LDS_j RT - LSS_i ST)}{2}$
12. $p_{si \rightarrow sj} DR = \sum_i^j \frac{No. of PP Recieved_{s_j}}{No. of PP Sent_{s_i}}$
13. $p_{si \rightarrow sj} rC = C_{si \rightarrow sj} - \sum_i^j \frac{\rho[t] - \rho[t-1]}{\delta}$
14. $PSM = \sum_i^j ((\min(p_{si \rightarrow sj} D)) + (\min(p_{si \rightarrow sj} DR)) + (\max(p_{si \rightarrow sj} rC)))$
15. // Evaluate the cost and append to each $p_{ij} \in P$
16. Select the Max (PSM)
17. Install the rule in $Flow_{table}$
18. **End For**
19. **Else** $T_f \in LC_{risk}$ **Then**
20. **For Each** $p_{ij} \in P$
21. Select a path using Dijkstra.
22. Install the rule in $Flow_{table}$
23. **End For**
24. **End IF**
25. **End**

4. Experimentation Setup to Validate APSAF

The experiment is conducted on a simulated environment on an Intel(R) Core (TM) i7-10750H CPU @ 2.60 GHz 2.59 GHz and 16.0 GB memory. Figure 5 illustrates the basic architecture of the simulation environment setup and how the various tools used for the experiment interact together. The tools comprised the controller at the control plane (CP), communicating with the data plane (DP) switches via NBi using OpenFlow protocol. A traffic generation tool is used to replicate the workload of internet applications for the proposed APSAF at the application layer (AP) to select a suitable path to route the traffic accordingly. See Section 4.1 for more information on these tools.

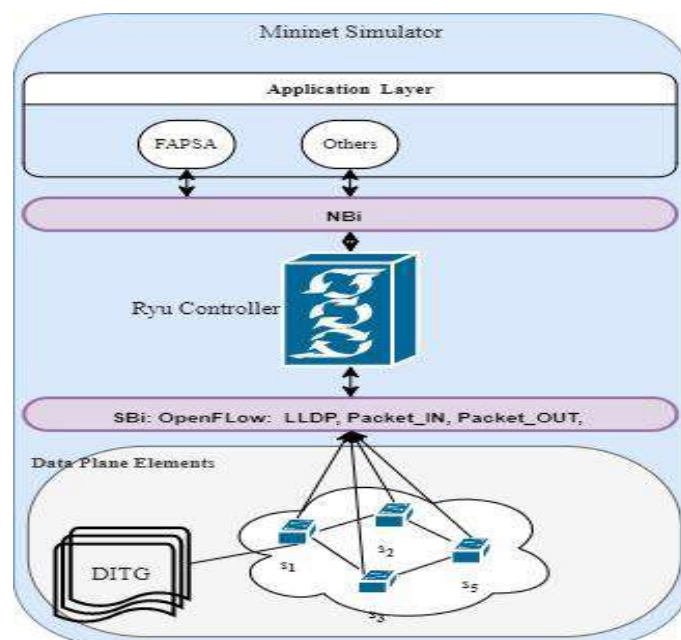


Figure 5. The architecture of simulation environment.

4.1. The Experimentation Tools and Data Traffic Model

As summarized in Table 2, a Ryu SDN controller [59] is configured on Oracle virtual machine (VM) version 6.1.18 with Ubuntu (64bit) to run the APSAF algorithm. FARSA selects paths to route traffic from a European Reference Network (ERnet) topology, which has 37 nodes and 57 edges with undirected links [38]. The Ernet topology is created using a script written in Python. Mininet version 2.3.0 [60] is employed to emulate the topology. All the Data Plane (DP) switches on the topology are designed with software such as OpenVswitch 3.0.1 (OVS) [61,62] on the emulator to interact with the Ryu controller using OpenFlow protocol v1.5.1 [58]. D-ITG utility [63] generates different quantities of TCP and UDP traffic flows for the experiment. The study models the traffic to follow poison traffic distribution in terms of Packet Inter Departure Time (PIDT) and Size (PS).

Table 2. Experimentation Tools and Data Traffic Model.

Experimentation Tools	Control Plane	Data Plane	Northbound	Emulator	Flat Form	Environment	Traffic Generator
	Ryu Controller	OpenVswitch	OpenFlow	Mininetv2.2.2	Ubuntu (64 bit)	OracleVM6.1.18	D-ITG
Traffic Parameters	Traffic Type	Inter Departure Time (IDT)	Size	Values	Control Utility		
	TCP, UDP	Constant and Poison distribution pattern		0.1–10, 15, 20	TCLink		

4.2. Performance Evaluation of APSAF

Selecting a path suitable to route-susceptible congestion traffic flows such as EF without hurting MF, which are delay sensitive, is significant in meeting the QoS of networks. To this end, APSAF is proposed to provide this solution. An experiment is conducted to explore how some network performance metrics relevant to the algorithm's objective are met. Accordingly, this study adopts three main metrics, which comprise throughput [54], Packet Delivery Ratio (PDR) [55], and Path Load Ratio (LPR) [12], to carry out the performance evaluation. The Path Load Ratio (PLR) will provide information on the traffic load ratio on a path to the path capacity. The motive is to find out the performance benefits while using APSAF as the path selection method in SDN. Similarly, APSAF is validated through comparison with other existing path selection methods in SDN, such as [14], to assess its effectiveness and the improvement achieved. Finally, a discussion and analysis are provided in the following subsections.

4.2.1. Throughout

Throughput provides network performance information about the number of data packets effectively delivered at the destination host over a transmission period sent from a source host [64]. The metric is relevant in assessing path selection technique performance on how it responds to network-changing events such as traffic arrival rate or failure [12]. Figure 6 compares the throughputs achieved while selecting a path to route traffic with the Algorithm proposed in this work, APSAF, and other PSAs proposed in [8,10,12]. As can be seen from Figure 6, APSAF achieves 11.2%, 38.2%, and 57.6% improvement in terms of throughput compared to RPSO, FSL, and RPF, respectively. The result indicates that the adopted methodology of separating large flows from smaller flows to select paths according to their respective demands facilitates a better path selection decision. This analysis supports the idea [13] that amalgamating all flows on the same path, irrespective of their attributes, affects QoS, such as throughput. APSAF exhibits significant path selection decisions consistently compared to FSL and RPF because of the adopted methodology. RPSO, on the other hand, closely matches APSAF because it selects a path with few critical switches to route the traffic flows. The relation pattern between APSAF and RPSO suggests that combining the flow classification with a switch role in path selection decisions might improve the throughput.

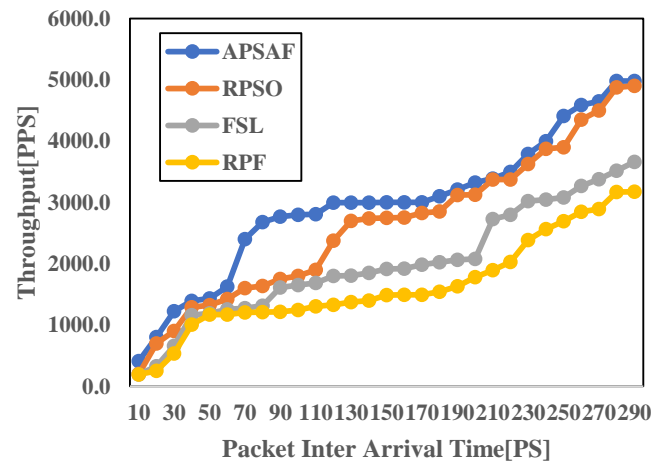


Figure 6. Throughput.

4.2.2. Packet Delivery Ratio (PDR)

The Packet Delivery Ratio (PDR) in network performance is defined as the ratio of successfully delivered data packets at the receiver to the total number of data packets transmitted at the source by all flows. The result in Figure 7 indicates how APSAF performed compared to RPSO [14], FSL, and RPF. The x and y axes in the graph represent the percentage of PDR and Packet Inter-Transfer Rate (PITR) per second. PDR is observed to be inversely proportional to the number of packets dropped. The packet drop might occur due to the path’s inability to accommodate anomalies and changes such as link failure, traffic variabilities, or burst flow arrival patterns in the network [64]. Thus, the metric is crucial in assessing the performance of PSAs because, under the same condition, the increase or decrease in PDR concerning PITR indicates an improved performance or otherwise. Therefore, different PITRs are generated to measure PDR in the experiments conducted. As shown in Figure 7, APSAF demonstrates a steady and better PDR compared to RPSO [64], FSL, and RPF by 3.3%, 31.8%, and 60.0%, respectively. The performance benefit is attributed to accurate path quality estimation of flow demand and subsequent rerouting of the packets through a path with adequate capacity to accommodate the traffic. Therefore, packet transmission may not be affected even when there is a change in the network, as observed from the significant improvement compared to FSL and RPF. However, based on the result, APSAF and RPSO move the head to shoulder as in the throughput. This behavior suggests that considering both flow classification and switch roles in path selection decisions might improve the PDR in a network. We plan to consider a practical implementation of this idea in our future research.

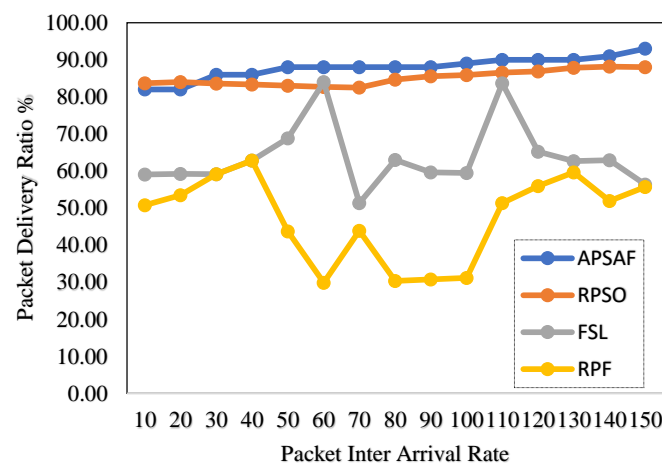


Figure 7. Packet Delivery Ratio.

4.2.3. Path Load Ratio (PLR)

The Path Load Ratio (PLR) is the ratio between the traffic load (in the form of the number of EF) on that path and the path capacity. This metric calculates the percentage of a path's capacity used. Therefore, for every given path $P\{p_{ij}, p_{ij} \dots n\}$, lower PLR values indicate a more optimal route decision for the current traffic flow [12]. In SDN, the OVS keeps track of the total number of bytes transmitted and received through each port. Accordingly, APSAF compiles the recorded statistics encompassing the start and end of the transmission to compute the bytes transmitted through the port during a particular period to derive the PLR. Next, we divide the period length by the path capacity to get the port's path utilization. Lastly, PLR equals the maximum path utilization across all ports in the network. This experiment examines the PLR's performance concerning variations in traffic flow arrivals. Thus, PLR is observed by gradually increasing the number of flows from 15 to 150,000,000. In Figure 8, the PLR of APSAF is comparable to that of RPSO within 5%; however, APSAF dramatically reduces the route load ratio by 26% and 44% relative to FSL and RPF, respectively. APSAF's performance gain can be traced back to its usage of flow classification and optimal path selection modules because the modules dynamically route traffic according to flow characteristics, with the ultimate goal of selecting the path with the highest PSM aggregate value.

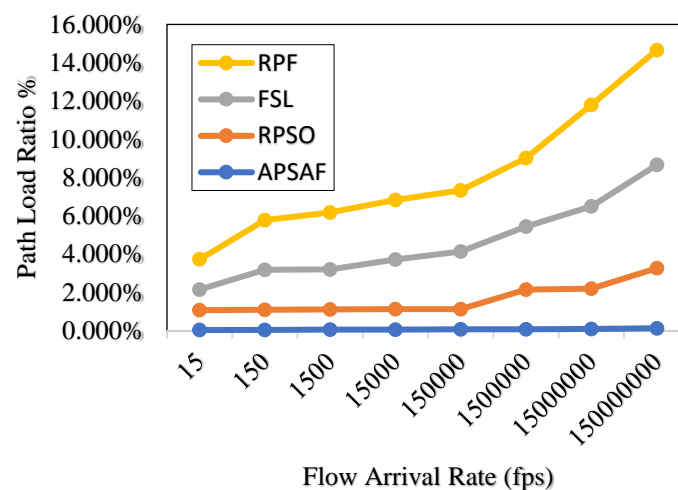


Figure 8. Packet Load Ratio.

4.2.4. Average Packet Delay

Figure 9a,b display the results for the average delay incurred while transmitting Elephant Flows and Mice Flows using APSAF and the result of APSAF in comparison with three other algorithms such as RPSO, RPF, and FSL while routing traffic coming at different flow arrival rates per second, respectively. From Figure 9a it can be observed that at the beginning of the transmission when the flow arrival rate is below 10 flows per second both traffic flows incurred delays below 1 ms. The delay increased when the arrival rate of flow increased beyond 20 flows per second. At this stage, the Elephant flow incurred higher delays than the Mice flow because of consideration of the path selection metrics. On the other hand, overall, APSAF achieves better average delay performance as compared to the benchmark algorithms. APSA achieves this result because it implements a thread monitoring mechanism to detect congestion-prone flows and reroutes them to a path with an appropriate capacity to accommodate them. Thus, it can guarantee the requirements of each flow category did not influence one another. As such, the Mice flow selects the shortest distance path to mitigate the effect of the delay.

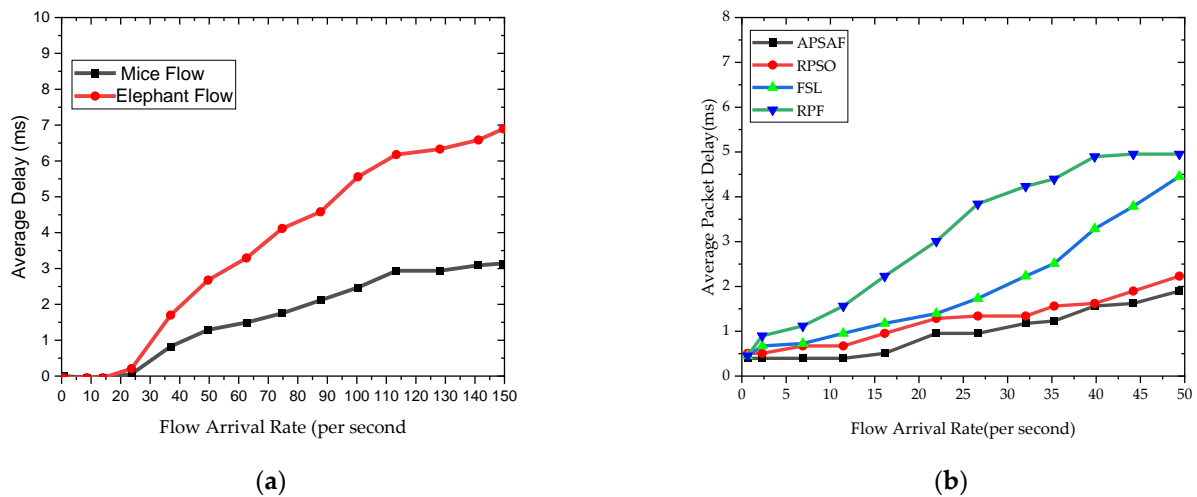


Figure 9. (a) Average Packet Delay; (b) Average Packet Delay.

4.2.5. Packet Loss

Figure 10a depicts the result of the packet loss rate experiments. By differentiating among flows and limiting the Mice Flow to a path whose selection is based on the shortest distance, the APSAF approach achieved the lowest packet loss rate for the transmission of Elephant Flows. In addition, APSAF, RPSO, RPF, and FSL are compared and the results of the experiment of the four strategies are displayed in Figure 10b. Since APSAF uses path selection measures that take its packet loss rate and delivery ratio into account when transmitting the Elephant Flows, it significantly lowers the packet loss rate. As shown, APSAF has the lowest average packet loss rate of the four routing algorithms due to its usage of the path suitability metric to guide the path selection decision. This is because path quality metrics such as delay, link residual capacity, and path delivery ratio are completely accounted for in the cost function. Meanwhile, it is noticed that when the arrival rate is quite low, there is no substantial difference in the outcomes of the three algorithms. However, when the flow arrival rate increases, APSAF’s packet loss rate is much lower than the benchmark algorithms.

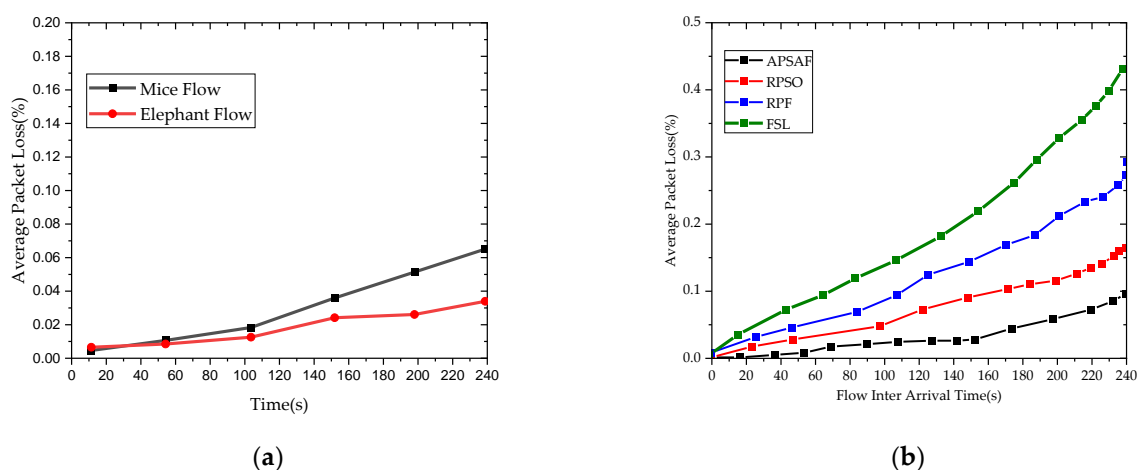


Figure 10. (a) Packet Loss. (a). Packet Loss.

5. Conclusions

Adaptive Path Selection Algorithm with Flow Classification for Software-Defined Networks is a novel approach to address the challenge of dynamic traffic management in SDN. This Algorithm utilizes the combination of flow classification and multiple path selection to improve the network’s ability to handle unpredictable and dynamic traffic.

The Algorithm has been implemented and evaluated in a laboratory testbed, and the results demonstrate its effectiveness in terms of increased network performance, reduced congestion, and improved quality of service. This research presents a route path selection technique based on flow classification to differentiate congestion-prone flows, such as EF, from hurting the majority of delay-sensitive MF by rerouting them on paths with appropriate capacity to avoid congestion and loss in SDN. A route with the highest quality metrics is chosen using path selection techniques. A composite path quality estimation vector has been designed to select a path based on the QoS needed by each traffic category, as determined by the flow classification phase. The flow classification phase makes it possible for the path selection phase to map flows differently. Based on the experiments, the proposed solution has reduced the Path Load Ratio (PLR) by 37% while improving the throughput and packet delivery ratio by 55.73% and 12.5%, compared to the benchmark works. APSAF envisages leveraging graph theory techniques to evaluate switch roles in the network to isolate critical switches along a selected path to route any traffic in future work. The aim is to minimize the controller's overhead during rule update operations in the event of any topology change. Thus, the comparison of flow characteristics, path quality, and switch role can all be taken into account simultaneously to arrive at a path selection decision. Similarly, to improve the flow classification and prioritization phase, it would be intriguing to employ machine learning techniques such as deep learning to forecast flow behavior based on flow history. In the future, the proposed method could be to incorporate deep learning techniques to further enhance the flow classification process and make the Algorithm more adaptive to changing network conditions. Another avenue for exploration could be to integrate the Algorithm with edge computing to better support distributed applications and services. Additionally, future work could also focus on extending the Algorithm to support multi-domain SDN and evaluate its performance in real-world network deployments.

Author Contributions: Conceptualization, M.N.Y. and B.I.; methodology, M.N.Y., K.b.A.B. and B.I.; validation, M.N.Y., K.b.A.B. and B.I.; writing—original draft preparation, M.N.Y.; writing—review and editing, A.H.O. and M.N.; supervision, K.b.A.B.; funding acquisition, A.H.O., M.N. and F.A.E. All authors have read and agreed to the published version of the manuscript.

Funding: King Abdulaziz University-Institutional Funding Program for Research and Development-Ministry of Education: IFPIP: 464-830-1443.

Data Availability Statement: Not applicable.

Acknowledgments: This research work was funded by Institutional Fund Projects under grant no. (IFPIP:464-830-1443). The authors gratefully acknowledge the technical and financial support provided by the Ministry of Education and King Abdulaziz University, DSR, Jeddah, Saudi Arabia.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviation

SDN	Software-Defined Networking
DP	Data Plane
CP	Control Plane
AP	Application Plane
NBi	Northbound Interface
SBi	Southbound Interface
NOS	Network's Operating System
POF	Protocol Oblivious Forwarding
OVSDB	Open vSwitch Database
ForCES	Forwarding Control Elements
LLDP	Link Layer Discovery Protocol

PP	Prove Packet
OUI	Organizational Unique Identifier
TTL	Time To Live
TLV	Type Length Value
LSS	Link Source Switch
LDS	Link Destination Switch
OXS	OpenFlow Extensible Stat
QoS	Quality of Service
SLA	Service-Level Agreement
TCP	Transmission Control Protocol
UDP	User Datagram Protocol,
ICMP	Internet Control Message Protocol
D-ITG	Distributed Internet Traffic Generator
Tx	Transmission
Rx	Receive
SPP	Special Ping Packet
RTT	Round-Trip Time
EF	Elephant Flow
LLF	Long-Lived Flow
MF	Mice Flow
SLF	Long-Lived Flow
ESD	Edge Switch Detection
EHD	End-Host Detection
STD	Switch Trigger Detection
IoT	Internet of Things
SPOF	Single Point Failure
LP	Linear Programming
ILP	Integer Linear Programming
PSD	Path Selection Decision
PSA	Path Selection Algorithm
PSM	Path Selection Method
LLP	Least Loaded Path
ECMP	Equal Cost Multiple Path
DMSP	Dynamic Multipath Scheduling Protocol
GOMR	Globally Optimized Multipath Routing
QSMPS	QoS Multipath Selection Scheme
RPSO	Routh Selection Optimization
FSL	Flow Setup latency
RPF	Reliable Path Finder
APSAF	Adaptive Path Selection Algorithm with Flow Classification
DPED	Data Plane Elements Discovery
FCP	Flow Classification Phase
PIDT	Packet Inter Departure Time
PS	Packet Size
PSM	Path Suitability Metric
PSSP	Path Screening and Selection Phase
ERnet	European Reference Network
OVS	OpenVswitch
PDR	Path Delivery Ratio
PLR	Path Load Ratio
PITR	Packet Inter-Transfer Rate

References

1. Isyaku, B.; Zahid, M.S.M.; Kamat, M.B.; Bakar, K.A.; Ghaleb, F.A. Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey. *Future Internet* **2020**, *12*, 147. [[CrossRef](#)]
2. 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](#)]

3. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [[CrossRef](#)]
4. Yusuf, M.N.; Bakar, K.B.A.; Isyaku, B.; Mukhlif, F. Distributed Controller Placement in Software-Defined Networks with Consistency and Interoperability Problems. *J. Electr. Comput. Eng.* **2023**, *2023*, 6466996. [[CrossRef](#)]
5. Muthanna, M.S.A.; Alkanhel, R.; Muthanna, A.; Rafiq, A.; Abdullah, W.A.M. Towards SDN-Enabled, Intelligent Intrusion Detection System for Internet of Things (IoT). *IEEE Access* **2022**, *10*, 22756–22768. [[CrossRef](#)]
6. Al Razib, M.; Javeed, D.; Khan, M.T.; Alkanhel, R.; Muthanna, M.S.A. Cyber Threats Detection in Smart Environments Using SDN-Enabled DNN-LSTM Hybrid Framework. *IEEE Access* **2022**, *10*, 53015–53026. [[CrossRef](#)]
7. Lin, C.Y.; Chen, C.; Chang, J.W.; Chu, Y.H. Elephant Flow Detection in Datacenters Using OpenFlow-Based Hierarchical Statistics Pulling. In Proceedings of the 2014 IEEE Global Communications Conference, Austin, TX, USA, 8–12 December 2014; pp. 2264–2269. [[CrossRef](#)]
8. Malik, A.; Aziz, B.; Bader-El-Den, M. Finding Most Reliable Paths for Software Defined Networks. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 1309–1314. [[CrossRef](#)]
9. Jamali, S.; Badirzadeh, A.; Siapoush, M.S. On the use of the genetic programming for balanced load distribution in software-defined networks. *Digit. Commun. Netw.* **2019**, *5*, 288–296. [[CrossRef](#)]
10. Khalili, R.; Despotovic, Z.; Hecker, A. Flow Setup Latency in SDN Networks. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 2631–2639. [[CrossRef](#)]
11. Chooprateep, A.; Somchit, Y. Video Path Selection for Traffic Engineering in SDN. In Proceedings of the 2019 11th International Conference on Information Technology and Electrical Engineering (ICITEE), Pattaya, Thailand, 10–11 October 2019. [[CrossRef](#)]
12. Isyaku, B.; Bakar, K.A.; Zahid, M.S.M.; Alkhamash, E.H.; Saeed, F.; Ghaleb, F.A. Route path selection optimization scheme based link quality estimation and critical switch awareness for software defined networks. *Appl. Sci.* **2021**, *11*, 9100. [[CrossRef](#)]
13. Qi, W.; Song, Q.; Kong, X.; Guo, L. A traffic-differentiated routing Algorithm in Flying Ad Hoc Sensor Networks with SDN cluster controllers. *J. Frankl. Inst.* **2019**, *356*, 766–790. [[CrossRef](#)]
14. Jiawei, W.; Xiuquan, Q.; Chen, J. PDMR: Priority-based dynamic multi-path routing Algorithm for a software defined network. *IET Commun.* **2019**, *13*, 179–185. [[CrossRef](#)]
15. Gotani, K.; Takahira, H.; Hata, M.; Guillen, L.; Izumi, S.; Abe, T.; Suganuma, T. Design of an SDN Control Method Considering the Path Switching Time under Disaster Situations. In Proceedings of the 2018 5th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM), Sendai, Japan, 4–7 December 2018; pp. 1–4. [[CrossRef](#)]
16. Yu, C.; Zhao, Z.; Zhou, Y.; Zhang, H. Intelligent Optimizing Scheme for Load Balancing in Software Defined Networks. In Proceedings of the 2017 IEEE 85th Vehicular Technology Conference (VTC Spring), Sydney, Australia, 4–7 June 2017; pp. 5–9. [[CrossRef](#)]
17. Rashid, J.A. Sorted-GFF: An efficient large flows placing mechanism in software defined network datacenter. *Karbala Int. J. Mod. Sci.* **2018**, *4*, 313–331. [[CrossRef](#)]
18. Hao, J.; Shi, Y.; Sun, H.; Sheng, M.; Li, J. Rerouting Based Congestion Control in Data Center Networks. In Proceedings of the 2019 IEEE International Conference on Communications Workshops (ICC Workshops), Shanghai, China, 20–24 May 2019. [[CrossRef](#)]
19. Qin, K.; Fu, B.; Chen, P.; Huang, J. MCRA : Multicost Rerouting Algorithm in SDN. *J. Adv. Comput. Intell. Inform.* **2020**, *24*, 728–737. [[CrossRef](#)]
20. Lan, K.C.; Heidemann, J. A measurement study of correlations of Internet flow characteristics. *Comput. Netw.* **2006**, *50*, 46–62. [[CrossRef](#)]
21. Wang, H.; Xu, H.; Qian, C.; Ge, J.; Liu, J.; Huang, H. PrePass: Load balancing with data plane resource constraints using commodity SDN switches. *Comput. Netw.* **2020**, *178*, 107339. [[CrossRef](#)]
22. Al-Fares, M.; Radhakrishnan, S.; Raghavan, B.; Huang, N.; Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2010), San Jose, CA, USA, 28–30 April 2010; pp. 281–295.
23. Ongaro, F.; Cerqueira, E.; Foschini, L.; Corradi, A.; Gerla, M. Enhancing the Quality Level Support for Real-Time Multimedia Applications in Software-Defined Networks. In Proceedings of the 2015 International Conference on Computing, Networking and Communications (ICNC), Garden Grove, CA, USA, 16–19 February 2015; pp. 505–509. [[CrossRef](#)]
24. Sahhaf, S.; Tavernier, W.; Colle, D.; Pickavet, M. Adaptive and reliable multipath provisioning for media transfer in SDN-based overlay networks. *Comput. Commun.* **2017**, *106*, 107–116. [[CrossRef](#)]
25. Guo, Z.; Xu, Y.; Liu, R.; Gushchin, A.; Chen, K.-Y.; Walid, A.; Chao, H.J. Balancing flow table occupancy and link utilization in software-defined networks. *Futur. Gener. Comput. Syst.* **2018**, *89*, 213–223. [[CrossRef](#)]
26. Sminesh, C.N.; Kanaga, E.G.M.; Ranjitha, K. A proactive flow admission and re-routing scheme for load balancing and mitigation of congestion propagation in SDN data plane. *Int. J. Comput. Netw. Commun.* **2018**, *10*, 117–134. [[CrossRef](#)]
27. Guo, Y.; Luo, H.; Wang, Z.; Yin, X.; Wu, J. Routing optimization with path cardinality constraints in a hybrid SDN. *Comput. Commun.* **2021**, *165*, 112–121. [[CrossRef](#)]
28. Benson, T.; Anand, A.; Akella, A.; Zhang, M. MicroTE: Fine grained traffic engineering for data centers. In Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies, Tokyo, Japan, 6–9 December 2011. [[CrossRef](#)]

29. Curtis, A.R.; Kim, W.; Yalagandula, P. Mahout: Low-Overhead Datacenter Traffic Management Using End-Host-Based Elephant Detection. In Proceedings of the 2011 Proceedings IEEE INFOCOM, Shanghai, China, 10–15 April 2011; pp. 1629–1637. [[CrossRef](#)]
30. Cheung, C.M.; Leung, K.C. DFFR: A flow-based approach for distributed load balancing in Data Center Networks. *Comput. Commun.* **2018**, *116*, 1–8. [[CrossRef](#)]
31. Rottenstreich, O.; Kanizo, Y.; Kaplan, H.; Rexford, J. Accurate Traffic Splitting on SDN Switches. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 2190–2201. [[CrossRef](#)]
32. Rifai, M.; Huin, N.; Caillouet, C.; Giroire, F.; Lopez-Pacheco, D.; Moulhierac, J.; Urvoy-Keller, G. Too Many SDN Rules ? COMPRESS them with MINNIE To cite this version : HAL Id : Hal-01203020 Too Many SDN Rules ? Compress them with M INNIE. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015.
33. Braun, W.; Menth, M. Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking. In Proceedings of the 2014 Third European Workshop on Software Defined Networks, Budapest, Hungary, 1–3 September 2014; Volume 12307, pp. 25–30. [[CrossRef](#)]
34. Kannan, K.; Banerjee, S. Compact TCAM: Flow entry compaction in TCAM for power aware SDN. *Lect. Notes Comput. Sci.* **2013**, *7730*, 439–444. [[CrossRef](#)]
35. Banerjee, S.; Kannan, K. Tag-In-Tag: Efficient flow table management in SDN switches. In Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop, Rio de Janeiro, Brazil, 17–21 November 2014; pp. 109–117. [[CrossRef](#)]
36. Hopps, C. *Analysis of an Equal-Cost Multi-Path Algorithm*; The Internet Society: Reston, VA, USA, 2000; pp. 130–139.
37. Curtis, A.R.; Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Banerjee, S. DevoFlow: Scaling flow management for high-performance networks. *Comput. Commun. Rev.* **2011**, *41*, 254–265. [[CrossRef](#)]
38. Tang, F.; Zhang, H.; Yang, L.T.; Chen, L. Elephant Flow Detection and Differentiated Scheduling with Efficient Sampling and Classification. *IEEE Trans. Cloud Comput.* **2019**, *9*, 1022–1036. [[CrossRef](#)]
39. Hsu, K.F.; Tammana, P.; Beckett, R.; Chen, A.; Rexford, J.; Walker, D. Adaptive weighted traffic splitting in programmable data planes. In Proceedings of the SOSR 2020—Proceedings of the Symposium on SDN Research, San Jose, CA, USA, 3 March 2020; pp. 103–109. [[CrossRef](#)]
40. Hussain, S.A.; Akbar, S.; Raza, I. A Dynamic Multipath Scheduling Protocol (DMSP) for Full Performance Isolation of Links in Software Defined Networking (SDN). In Proceedings of the 2017 2nd Workshop on Recent Trends in Telecommunications Research (RTTR), Palmerston North, New Zealand, 10 February 2017. [[CrossRef](#)]
41. Farrugia, N.; Buttigieg, V.; Briffa, J.A. A Globally Optimised Multipath Routing Algorithm Using SDN. In Proceedings of the 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 19–22 February 2018; pp. 1–8. [[CrossRef](#)]
42. Kakahama, H.K.; Taha, M. Adaptive Software-defined Network Controller for Multipath Routing based on Reduction of Time. *UHD J. Sci. Technol.* **2020**, *4*, 107–116. [[CrossRef](#)]
43. Yan, J.; Zhang, H.; Shuai, Q.; Liu, B.; Guo, X. HiQoS: An SDN-based multipath QoS solution. *China Commun.* **2015**, *12*, 123–133. [[CrossRef](#)]
44. Luo, M.; Zeng, Y.; Li, J.; Chou, W. An adaptive multi-path computation framework for centrally controlled networks. *Comput. Netw.* **2015**, *83*, 30–44. [[CrossRef](#)]
45. Jin, H.; Yang, G.; Yu, B.Y.; Yoo, C. TALON: Tenant Throughput Allocation Through Traffic Load-Balancing in Virtualized Software-Defined Networks. In Proceedings of the 2019 International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia, 9–11 January 2019; pp. 233–238. [[CrossRef](#)]
46. He, K.; Rozner, E.; Agarwal, K.; Felter, W.; Carter, J.; Akella, A. Presto. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 465–478. [[CrossRef](#)]
47. Yoo, Y.; Yang, G.; Lee, J.; Shin, C.; Kim, H.; Yoo, C. TeaVisor: Network Hypervisor for Bandwidth Isolation in SDN-NV. In *IEEE Transactions on Cloud Computing*; IEEE: Piscataway, NJ, USA, 2022. [[CrossRef](#)]
48. Saha, N.; Misra, S.; Bera, S. Sway: Traffic-Aware QoS Routing in Software-Defined IoT. In *IEEE Transactions on Emerging Topics in Computing*; IEEE: Piscataway, NJ, USA, 2018.
49. Prete, L.R.; Shinoda, A.A.; Schweitzer, C.M.; De Oliveira, R.L.S. Simulation in an SDN Network Scenario Using the POX Controller. In Proceedings of the 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), Bogota, Colombia, 4–6 June 2014. [[CrossRef](#)]
50. Botta, A.; Dainotti, A.; Pescapé, A. A tool for the generation of realistic network workload for emerging networking scenarios. *Comput. Netw.* **2012**, *56*, 3531–3547. [[CrossRef](#)]
51. Perner, C.; Carle, G. Comparison of Optimization Goals for Resilient Routing. In Proceedings of the 2019 IEEE International Conference on Communications Workshops (ICC Workshops), Shanghai, China, 20–24 May 2019; pp. 1–6. [[CrossRef](#)]
52. Malik, A.; de Fréin, R.; Aziz, B. Rapid restoration techniques for software-defined networks. *Appl. Sci.* **2020**, *10*, 3411. [[CrossRef](#)]
53. Ravuri, H.K.; Vega, M.T.; Wauters, T.; Da, B.; Clemm, A.; De Turck, F. An Experimental Evaluation of Flow Setup Latency in Distributed Software Defined Networks. In Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019; pp. 432–437. [[CrossRef](#)]
54. Isyaku, B.; Bakar, K.A.; Zahid, M.S.M.; Yusuf, M.N. Adaptive and Hybrid Idle–Hard Timeout Allocation and Flow Eviction Mechanism Considering Traffic Characteristics. *Electronics* **2020**, *9*, 1983. [[CrossRef](#)]

55. Dijkstra, E.W. Dijkstra.Pptx. *Numer. Math.* **1959**, *271*, 269–271. [[CrossRef](#)]
56. Yen, J.Y. Finding the K Shortest Loopless Paths in a Network. *Manag. Sci.* **1971**, *17*, 712–716. [[CrossRef](#)]
57. Rangkyuty, M.F.; Muslim, R.; Ahmad, T.; Al-Hooti, M.H.A. Path Selection in Software Defined Network Data Plane Using Least Loaded Path. In Proceedings of the 2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Depok, Indonesia, 17–18 October 2020; pp. 135–140. [[CrossRef](#)]
58. Morales, L.V.; Murillo, A.F.; Rueda, S.J. Extending the Floodlight Controller. In Proceedings of the 2015 IEEE 14th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 28–30 September 2015; pp. 126–133. [[CrossRef](#)]
59. Gao, K.; Xu, C.; Qin, J.; Yang, S.; Zhong, L.; Muntean, G.M. QoS-driven Path Selection for MPTCP: A Scalable SDN-assisted Approach. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019. [[CrossRef](#)]
60. Xu, X.; Hu, L.; Lin, H.; Fan, Z. An Adaptive Flow Table Adjustment Algorithm for SDN. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1779–1784. [[CrossRef](#)]
61. Maximets, I. [ovs-announce] Open vSwitch 3. 2022. Available online: <https://mail.openvswitch.org/pipermail/ovs-announce/2022-October/000300.html> (accessed on 11 July 2022).
62. Botta, A.; De Donato, W.; Dainotti, A.; Avallone, S.; Pescap, A. *D-ITG § VERSION § Manual*; University of Naples Federico II: Naples, Italy, 2019; pp. 1–35.
63. Afek, Y.; Bremner-Barr, A.; Feibish, S.L.; Schiff, L. Detecting heavy flows in the SDN match and action model. *Comput. Networks* **2018**, *136*, 1–12. [[CrossRef](#)]
64. Liao, L.; Leung, V.C.M.; Chen, M. An Efficient and Accurate Link Latency Monitoring Method for Low-Latency Software-Defined Networks. *IEEE Trans. Instrum. Meas.* **2018**, *68*, 377–391. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.