Research Article

Tameem Hameed Obaida* and Hanan Abbas Salman

A novel method to find the best path in SDN using firefly algorithm

https://doi.org/10.1515/jisys-2022-0063
received February 16, 2022; accepted May 14, 2022

Abstract: Over the previous three decades, the area of computer networks has progressed significantly, from traditional static networks to dynamically designed architecture. The primary purpose of software-defined networking (SDN) is to create an open, programmable network. Conventional network devices, such as routers and switches, may make routing decisions and forward packets; however, SDN divides these components into the Data plane and the Control plane by splitting distinct features away. As a result, switches can only forward packets and cannot make routing decisions; the controller makes routing decisions. OpenFlow is the communication interface between the switches and the controller. It is a protocol that allows the controller to identify the network packet's path across the switches. This project uses the SDN environment to implement the firefly optimization algorithm to determine the shortest path between two nodes in a network. The firefly optimization algorithm was implemented using Ryu control. The results reveal that using the firefly optimization algorithm improves the selected short path between the source and destination.

Keywords: Mininet, firefly algorithm, path selection, software-defined networking, Ryu control

1 Introduction

Optimal path selection necessitates continuous evaluation to ensure that the topology's linkages are promising quality pathways; besides, greater results are more likely to be used [1]. However, dynamic path ranking utilizing end-to-end active measures in large-scale networks is not scalable and efficient [2]. Because software-defined networking (SDN) controllers have a global view of the topology and access to a significant amount and diverse network data, a data-driven approach is worth investigating. According to specific research, network controller data may be used to learn correlations, enhancing network performance and resource allocation [3]. As a result, it is worth looking at a data-driven solution that uses data from controllers to guide the choice of path [1].

SDN brings up new options for Internet packet forwarding and flexible routing [4]. Switch control planes and forwarding planes are separated by SDN, enabling the establishment of forwarding tables to be done remotely and dynamically; as a result, SDN accomplishes at least three key interdomain traffic engineering objectives [5]: packet forwarding based on a variety of header attributes, remote forwarding rule setup, and dynamic/programmatic packet forwarding rule configuration. The Firefly algorithm (FA) is a meta-heuristic program that simulates firefly brood parasitism. The firefly's glow symbolizes the potential solutions. The algorithm gradually replaces the wrong solutions with new and better ones. The FA may be used in various domains, including neural networks, job scheduling, and so on.

* **Corresponding author: Tameem Hameed Obaida,** Department of Computer Systems Techniques, Al-Furat Al-Awsat Technical University, Najaf Technical Institute, Najaf, Iraq, e-mail: tameem.daham@atu.edu.iq
Hanan Abbas Salman: Department of Computer Systems Techniques, Al-Furat Al-Awsat Technical University, Najaf Technical Institute, Najaf, Iraq, e-mail: hananabbas@atu.edu.iq

The ability to locate other pathways and dynamically build routes depending on path attributes is examined in this research to see if it may assist in increasing link usage and network performance. This article explains how Ubuntu may increase bandwidth usage and decrease latencies by employing SDN-based traffic engineering and network measurements to accomplish dynamic path selection. The research also considers leveraging network data collected from probes between switches and an SDN controller to apply the FA to path selection. Mininet is used to test an SDN-based network emulator, which uses fireflies to disperse traffic over many forwarding connections to increase throughput and lower latency.

The following are the work's key contributions:

- A description of the methods for calculating a packet's predicted path in a given setup, recording the authentic way, and comparing the two pathways to discover a point of divergence.
- The implementation of a system prototype.
- An experiment involving the insertion of persistent and transient defects into network components

The remainder of this article is structured in the following manner. Section 2 gives the related study on power conservation and load balancing. We construct an SDN-based system and introduce its workflow in Section 3. Section 4 outlines the model and formulates the problem, followed by Section V, which presents our approach. Section 6 discusses performance evaluation. Finally, Section 7 brings this article to a close and offers new areas for future research.

2 Related work

Our strategy for achieving the objectives outlined in Section 2 is based on concepts offered in prior work in SDN network testing, verification, and debugging. Testing and verification methods aim to validate programs in terms of previously defined target invariants. Approaches to debugging are adapted to fix problems as they arise. We discuss relevant route optimization research. Routing optimization techniques for typically dispersed networks, mostly Optimization of open shortest path first (OSPF) link weights, is the main emphasis. Fortz and and Thorup [6] demonstrated a system for intra-domain routing optimization based on IP. They used a revised tabu search heuristic method to find the best OSPF weight setting. Ericsson et al. [7] presented a genetic approach to improve OSPF weight setting, while Srivastava et al. [8] offered a Lagrangian relaxation-based technique to improve routing in conventional networks. On the other hand, traditional dispersed networks force traffic to follow the shortest channels and offer no routing flexibility.

With the introduction of SDN, network operators may more easily operate the flow routing in their networks and alter their path choice as needed. Google [9] and Microsoft [10] have previously created fully SDN-enabled Inter-datacenter (Inter-DC) networks that can attain near-perfect by resolving a set of linear programming issues. Previous research has concentrated on routing optimization in a complete SDN network, which cannot be directly applied to hybrid SDN networks. Agarwal et al. [11] first addressed the route optimization difficulties in a hybrid SDN. To improve the network's flow splitting ratio routing and flow balancing, they offered a fully polynomial time approximation scheme (FPTAS). Hu et al. [12] and Wang et al. [13] created FPTAS to optimize traffic flows in a hybrid SDN, similar to ref. [11]. Guo et al. [14] offered heuristic techniques that concurrently optimize OSPF weight and splitting ratio of SDN nodes to increase network performance and decrease the MLU of the hybrid SDN. Hong et al. [15] presented a gradual hybrid network deployment and routing optimization solution. The pick group table feature offers heuristic techniques to send streams to the path with the least amount of traffic or numerous paths with varied possibilities. Jin et al. [16] presented a network controller for hybrid networks that allows for unified, fine-grained routing management. Chu et al. [17] presented a method for quickly reacting to single-link failure events in a mixed network while avoiding congestion. However, in a hybrid SDN, these earlier routing optimization techniques allow flows to route on any path from sources to destinations, regardless of path cardinality limitations. Caria et al. [18] split OSPF domains into numerous sub-domains and implemented SDN at border routers to allow for fine-grained traffic control across subdomains.

He and Song [19] presented polynomial-time approximation approaches for traffic engineering issues in two-hybrid modes, with an approximation of $(1 + \omega)$. Xu et al. [20] optimized routing in a particular hybrid SDN situation. In a hybrid network with SDN switches added to a standard IP network, they maximized incremental SDN rollout and flow routing together. Xu et al. [21] investigated entire SDN networks with conventional switching and SDN switching. Fibbing is a method proposed by Vissicchio et al. [22] for centrally controlling link-state routing protocols by creating fictitious nodes to provide additional routing flexibility. These hybrid network options differ from the hybrid SDN scenario we explored [23,24]. Other researcher works on wireless network with small network are refs [25–27].

According to our deep investigations in the previous work, the literature review could not present a sophisticated method for selecting short path from source to destination using firefly on SDN environment. Therefore, constructing the Ryu control have high impact to discover all paths from source to destination from manual method to an intelligent and automated method. To achieve that, the proposed method will be able to work the different topologies and select short path. The novelty in the proposed method is how to use FA to select short path based on SDN environment.

3 Background theory

3.1 Mininet

Mininet is a network emulator that precisely simulates the operation and performance of any sort of forwarding element. SDN networks may be built to precise standards and tested on various network setups. We may migrate the SDN solution to an existing physical network after completing the testing on Mininet [28].

3.2 FA

Yang created the Firefly method, a metaheuristic algorithm, in 2008 to solve optimization difficulties [29,30]. The following three principles helped to shape FA's configuration:

The light of one firefly attracts another. The fireflies with greater brightness levels have a higher level of appeal to other fireflies, and the fireflies with lower brightness levels go to the fireflies with higher brightness levels. Yang was motivated to create the FA by the three behaviors of natural fireflies. The actions of the firefly and the creation of FA have a close relationship. In reality, the brightness of each firefly corresponding to each ideal solution will be determined by the fitness function of the optimal solutions. The search for and acquisition of other fireflies producing greater brightness levels by fireflies with darker brightness levels is analogous to freshly created solutions depending on old solutions with a superior fitness function. As a result, in the FA, any previous solution might be recreated multiple times depending on how brilliant it is in contrast to others. As a consequence of the fitness function comparison, just one new solution of every previous solution is maintained.

Assume that each answer (X_i) represents a firefly i position at the present iteration. The distance between the fireflies when the fitness function of solution i is higher than that of solution j, the following equation is used to determine i and j.

$$r_{i,j} = \sqrt{(X_i - X_j)^2}, \tag{1}$$

The revised distance is then used to compute a new attractiveness by substituting it with another (2). Then, corresponding to creating a new ith solution, a new location for the ith considered firefly may be calculated. The technique for creating a new solution is implemented in the following manner (3):

$$\beta = \beta_0 e^{-\gamma r_{i,j}^2}, \tag{2}$$

$$X_{i,j_{\text{new}}} = X_i + \beta \cdot \text{rand} \cdot \Delta X_{i,j} + \text{rand}, \tag{3}$$

where rand is a random integer given to the solution, and i and 0 are the attractiveness at zero distance, usually 1. X_j is a solution with a lower fitness function than X_i, and $X_{i,j}$ is a step size computed using the model below.

$$\Delta X_{i,j} = (X_j - X_i), \tag{4}$$

The equations (1)–(3) of the ith solution are determined until no more solutions with a lower fitness function exist. In conclusion, we can have one, more than one, or no new solution for each solution depending on the fitness comparison between solution and other solutions inside the current population. The statement may be described using the following term.

$$\begin{aligned} X = X_i, \quad & \text{if } X_i \text{ is } X_{\text{Gbest}}\ X_{i\text{Gbest}}^{\text{new}}, \\ & \text{if } X_j \text{ is } X_{\text{Gbest}}\ X_{i,j}^{\text{new}} \text{FT}_{\text{Gbest}}, \text{ otherwise}, \end{aligned} \tag{5}$$

If the considered solution i is the global best solution, no new solution will be developed for the first term in (5). In the second case, if the considered solution is the second-best solution, just one new solution, X_i Gbest^new, will be developed, and X_j will be the population's global best solution, X Guest. In other cases, it means that X_i is the third-best or worse than the third-best answer, and that even if it is the worst choice, there will be two $(N-1)$ for new $X_{i,j}$ new options. In this situation, the fitness function values will be used to compare the set of new solutions for solution i, and the best one with the lowest fitness FT Gbest will be preserved, while the others will be deleted. Algorithm 1 may express the primary stages of the FA, which are based on three concepts.

Algorithm 1: FA

1	Fitness function $f(x) = (x_1, x_2, x_3, ..., x_d)^T$
2	Initializing a population of n firefly, $x_i (i \le n)$
3	Randomly generate N initial solution
4	For iteration in MaxGen
5	Compute brightness i
6	Sort solution from min to max
7	For i in $n-1$
8	For j in $i+1$
9	If $j >$ i
10	Move firefly i towards firefly j
11	End if
12	End for
13	End for
14	Move firefly N, (x), randomly
15	End for
16	Final result output and presentation

The fitness function determines the best communication channel. The value of the fitness function is affected by delay and distance. A path with the fewest node and the shortest distance is chosen as the best way to communicate.

4 Proposed model

The suggested multipath selection paradigm for routing in SDN is discussed in this section. Because of the advantages of the firefly search method, the suggested multipath routing selection surpasses the current

routing path. The FA is based on the firefly species' obligatory brood parasite habit combined with the typical flying behavior of birds and flies. The route is found using the FA. The benefits of the firefly search are considered while selecting the best path from the K paths identified during the path discovery phase. The following are the benefits of the FA: compared to other metaheuristic methods, it is more versatile and resilient for a wide range of optimization problems. It may readily be expanded to investigate multi-objective optimization problems with various constraints, including NP-hard problems. The suggested technique introduces a novel fitness function that considers several metrics such as distance and delay, resulting in improved performance with minimal latency. Figure 1 shows the overall block design of the proposed multipath selection method in SDN, based on the firefly search algorithm.

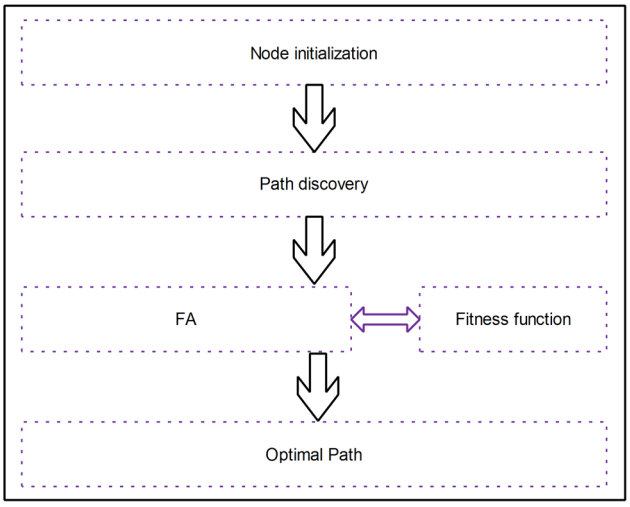


Figure 1: FA based on SDN.

This article presents a multipath technique for SDN networks based on the firefly search algorithm. The following three phases make up the implementation process:

- The supplied dynamic environment is used to initialize the nodes.
- The pathways from source to destination nodes are found.
- The fitness function and the FA are used to choose multipath.

Figure 1 demonstrates how the application plane uses the firefly search technique to determine the multipath topology and path status. The OpenFlow group table is installed in each path. The estimated multiple performance factors are assigned to the bucket value in the OpenFlow select group table, ensuring that network traffic is spread across all possible approaches based on the path value. Simulation findings demonstrate that this technique may increase multipath resource usage in SDN networks, dramatically improve traffic transmission efficiency, and accomplish multipath load balancing. In SDN, the firefly search method is presented in Figure 1.

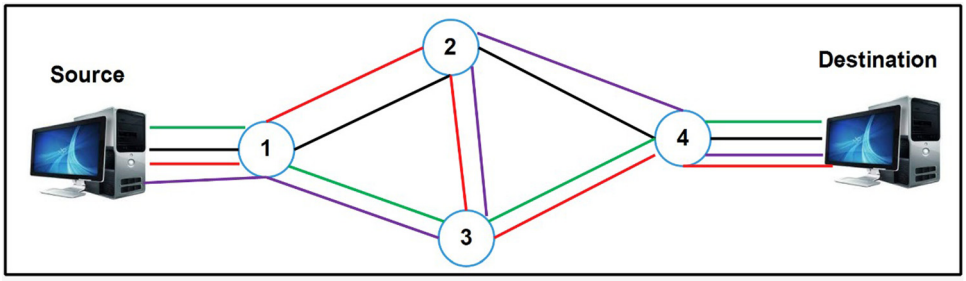
4.1 Node initialization

Because of its numerical control separation and programmability qualities, SDN technology allows the SDN controller to acquire and handle global network topology information. The switches module in the SDN controller implements the topology discovery and management mechanism by sending a packet out to the underlying network that contains the link layer discovery protocol (LLDP) (OpenFlow switch). After receiving the LLDP packet, the OpenFlow switch sends a packet to the controller carrying link information

between switches. The SDN controller then determines and maintains the network topology using the link discovery protocol's feedback information. A connection discovery approach like this consumes a lot of communication traffic and causes a delay. The SDN nodes are set up in the dynamic environment that has been selected. The nodes serve as both a router and a switch. In the network region, a link is established between the nodes. The nodes' coordinates are calculated, allowing them to be recognized in the future by their location and velocity. The network topology has m nodes, as defined by $1 < i < m$.

4.2 Path discovery

This work employs the firefly search method to accomplish multipath topology discovery, which reduces communication usage and reduces the cost of delay. Consider the network topology $G(H, S)$, where H denotes the number of hosts and S denotes the number of OpenFlow switches handled by the SDN controller. Let P represent the number of pathways that connect the source and destination nodes. $1 < j < P$ provides the answer. The firefly search path method is given in algorithm 1 to locate every possible multipath between two hosts. We can utilize the SDN controller to precisely get the whole network topology and link connection conditions by implementing this approach and giving usable information for future load balancing. Figure 2 depicts the path identified from source to destination nodes in a network topology with $m = 4$ nodes. The nodes are 1, 2, 3, and 4. The communication from source to destination is accomplished through numerous pathways. The path for data packet transmission is initially discovered depending on the nodes' connectivity. A link between Nodes 2 and 3 adds to the number of paths detected and the immediate connectivity between nodes in the immediate vicinity. Four pathways are found between the source and destination. 1–2–3–4, 1–3–2–4, 1–3–4, and 1–2–4 are the numbers. The suggested firefly search technique selects multipaths appropriate for communication between the hosts from the path identified utilizing an existing link between a node.



Color	Path	Cost
	1-2-4	3
	1-2-3-4	4
	1-3-2-4	4
	1-3-4	3

Figure 2: Path discovery.

Multiple pathways exist in the communication channel from source to destination nodes in network topology, depending on the link connected with the channel. The connection between the source and destination takes place across many pathways. K is the number of ways detected for L connections in the nodes linked with the communication channel from nodes between the source and destination.

4.3 Implementing the firefly search algorithm

The suggested firefly search method and the fitness function are used in this section to choose multipath. The firefly search algorithm's steps are outlined below.

Step 1: Let us suppose the host's nest is randomly initialized. The *K*-path detected is the size of a host nest. The goal is to select the best path among *K*-paths that have been found. Let us say the n host nest's starting population is:

$$f_y = f_1,\ f_2,\ f_3, \ldots,\ f_g, \tag{6}$$

In addition to the host, the population count is assumed to be *c*. For consecutive iterations, the count value gets incremented by 1.

Step 2: Using levy flight, generate a new solution (host nest) at random according to equation (1).

Step 3: Pick a nest at random. A solution is chosen randomly from the initialized host nest from equation (6). Let us call the chosen random solution f_y.

Step 4: Equation (7) used to feed fitness function in equation (3).

$$f_d = \min(p_d), \tag{7}$$

Assume f_d is a fractional produced solution and f_y is a solution picked at random from the nest as shown in equations (8) and (9).

$$\text{fitness}\ (f_d) = f_{d(\text{fit})}, \tag{8}$$

$$\text{fitness}\ (f_y) = f_{y(\text{fit})}, \tag{9}$$

where f_y is the solution picked at random from the nest and f_d is the fractional produced solution

Step 5: The worst nest solution is rejected based on the fitness function assessed in the worst-case rejection. The solution with the lowest fitness function is picked as the best. The worst-case rejection in the firefly search technique is determined by the discovery rate of the nest constructed using the proposed firefly search algorithm. The best solution is determined by equation (10).

$$\text{fit}(f_d) < \text{fit}(f_y), \tag{10}$$

Step 6: The design process includes iteration, ranking, and selection. Count *c* is increased until it reaches its maximum value. The best solution from each worst-case rejection is ranked depending on an introduced rank value. The solution with the highest rank chooses the optimal output solution for routing in the SDN.

5 Experiment and analysis

This part conducts simulation tests on this method to verify the previously presented model. The operating system in this experiment is Ubuntu 16.04, the network simulation software is Mininet, and the SDN controller is Ryu software. Mininet and Ryu are installed on a PC with a 2.60 GHz Intel i7 9750 CPU, 16GB of RAM, and a 64-bit Linux operating system. The Ryu controller runs on 64-bit Python 3.7 as its operating system. The suggested paradigm in this study is implemented using the Python programming language to create Ryu controller application files. Different topologies are employed in the experimental assessment to illustrate the performance of our suggested model. We will start with the environment setup in this part.

Extensive tests are then used to establish route selection and the deployment percentage of SDN nodes. After that, we show how well a suggested model performs in cost minimization using a predetermined path selection and the placement percentage of SDN nodes. Finally, we show how long the proposed model takes to compute. The criteria for firefly search are shown in Table 1.

Table 1: Firefly search parameters

Parameters	Value
Iteration	30
Firefly	15
Alpha	0.1
Beta	1.5
Param	0.25

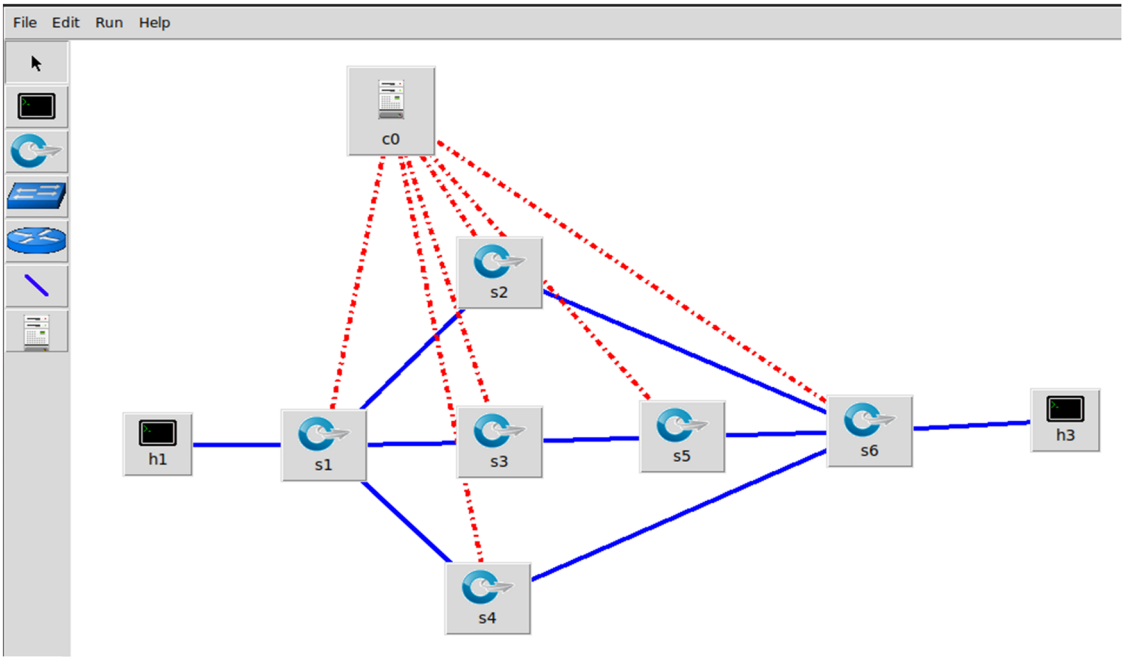


Figure 3: Network topology 1.

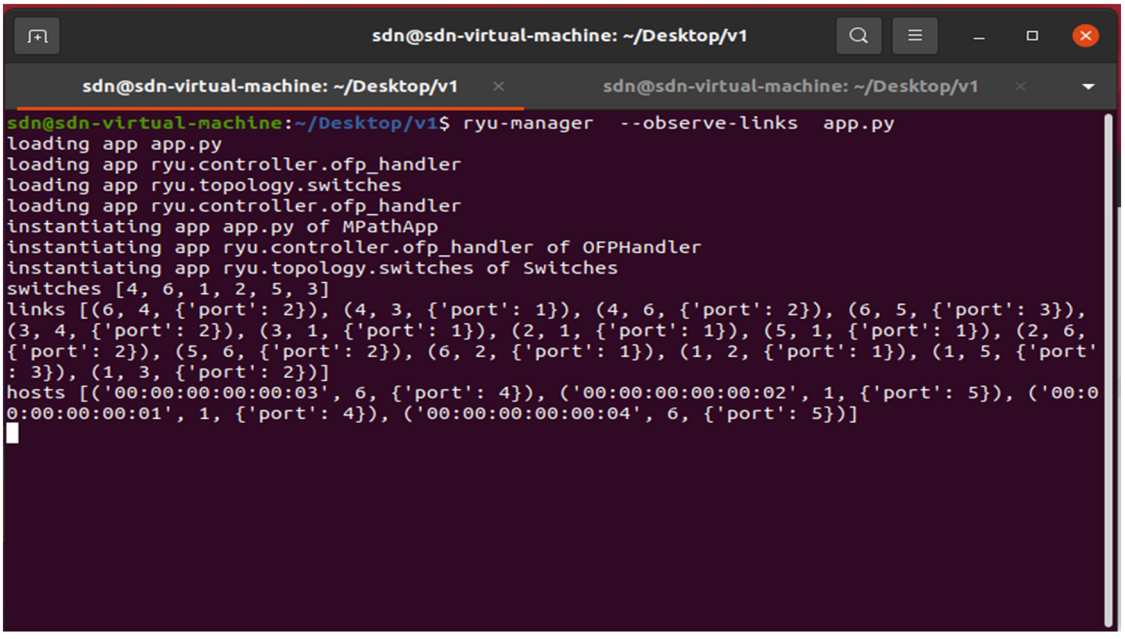


Figure 4: Running Ryu app.

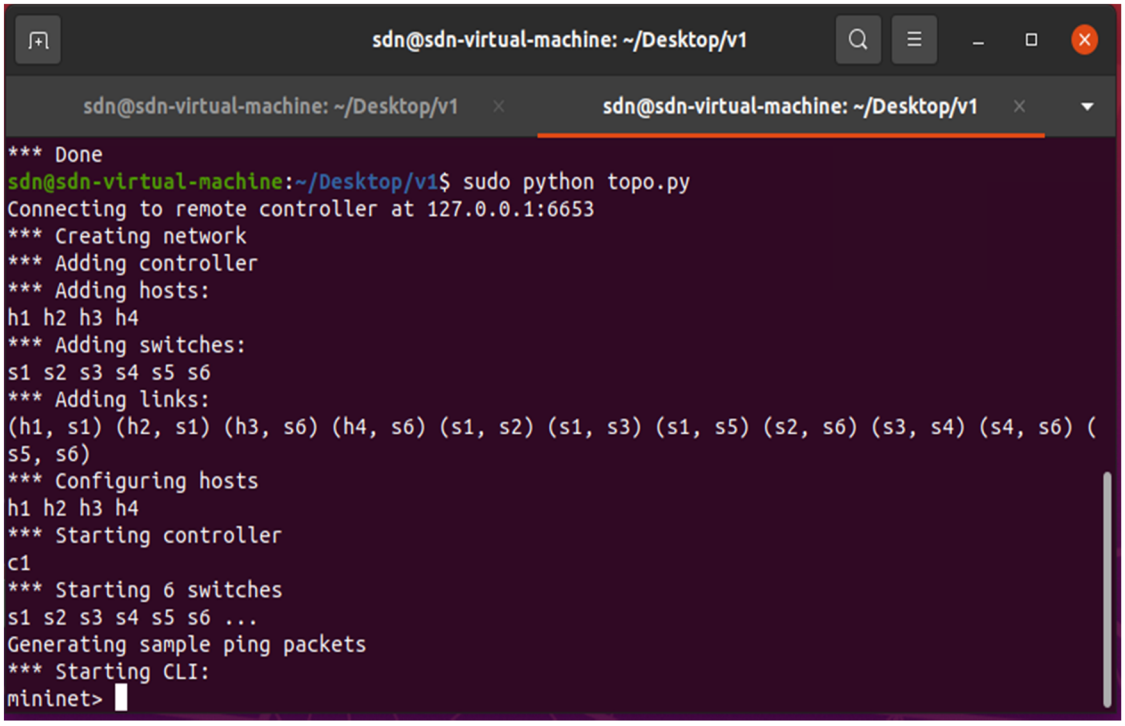


Figure 5: Adding of switches and links.

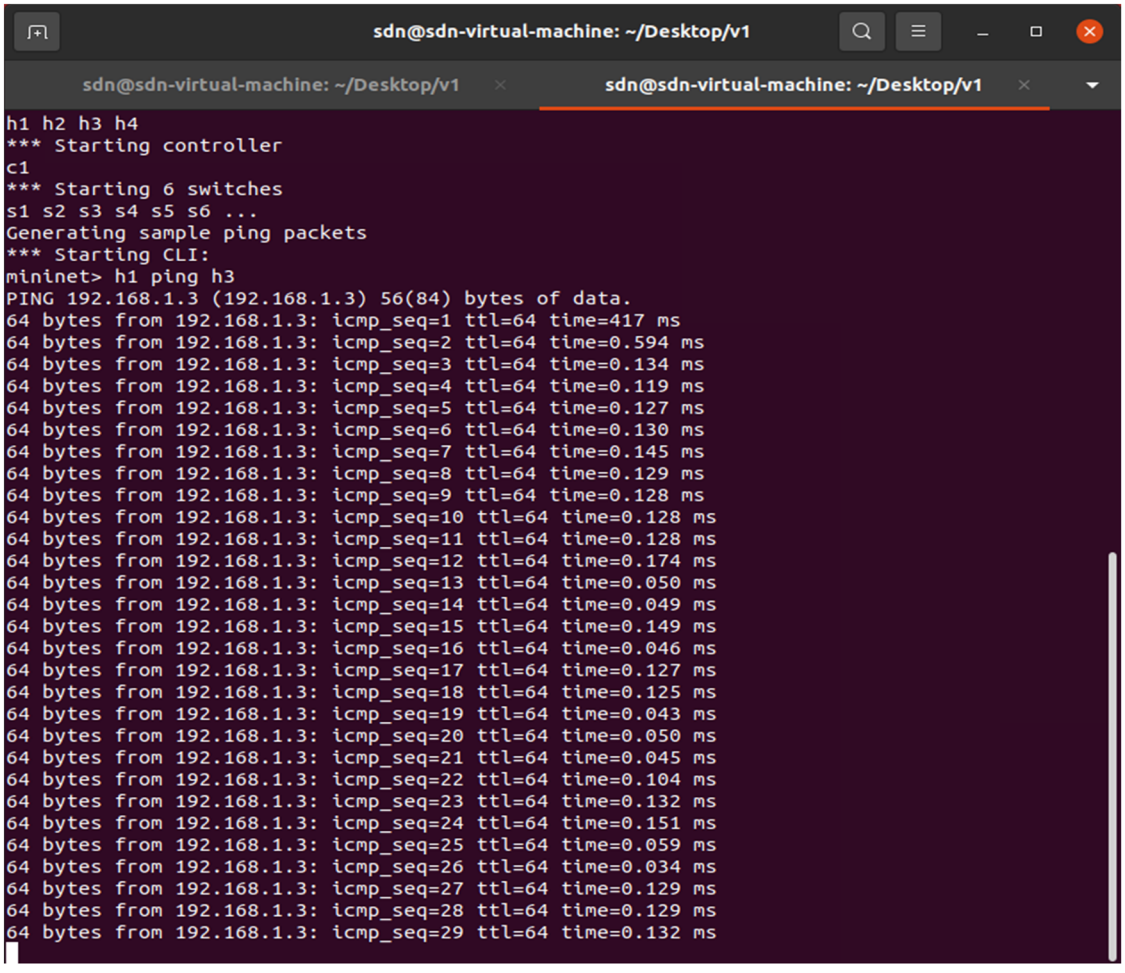


Figure 6: Ping from host 1 to host 3.

We must first establish the number and location of SDN nodes to acquire SDN. Mininet is used to build a network architecture with two hosts, six switches, and a controller, as illustrated in Figure 3. Traffic is routed with path selection limitations using the Firefly search algorithm.

When the SDN application "app" is started, it calls the launch function, which returns information about the Ryu controller and whether it is running (Figure 4). It also receives requests from a python script that constructs the network topology on port 6633 (which may be altered).

Figure 5 shows that anytime a switch is added to the network, it creates a "Switch connection event" after running a python script named "topo.py" containing the network's specifications. Additionally, "Received Link Event" and "Remove Link Event" are generated when a new link is added or withdrawn.

After all the switches and connections have been identified, packet transmission may begin. The firefly search method discovers the shortest channel for transmission when host "h1" pings "h3," as seen in Figure 6. One by one, the packets are transferred down the route. Initially, an ARP packet is sent to determine the IP addresses of all network devices involved and the flow tables of the switches.

Table 2 depicts the path that was built using the topological one. This scenario involves sending a message from Host 1 to Host 3 and deciding the best way to take it.

Table 2: Path discovery and selection

Path discovery	Hops	Path selection
s1–s2–s6	3	√
s1–s3–s5–s6	4	×
s1–s4–s6	3	√

Figure 7 depicts two hosts, h1 and h2, and six switches (s1, s2, s3, s4, s5, and s6) and a controller. In the diagram below, a new link is added between s2 and s3 and between s3 and s4.

Table 3 depicts the path chosen from topology 1 when it was formed. This scenario involves sending a message from Host 1 to Host 3 and determining the optimum way.

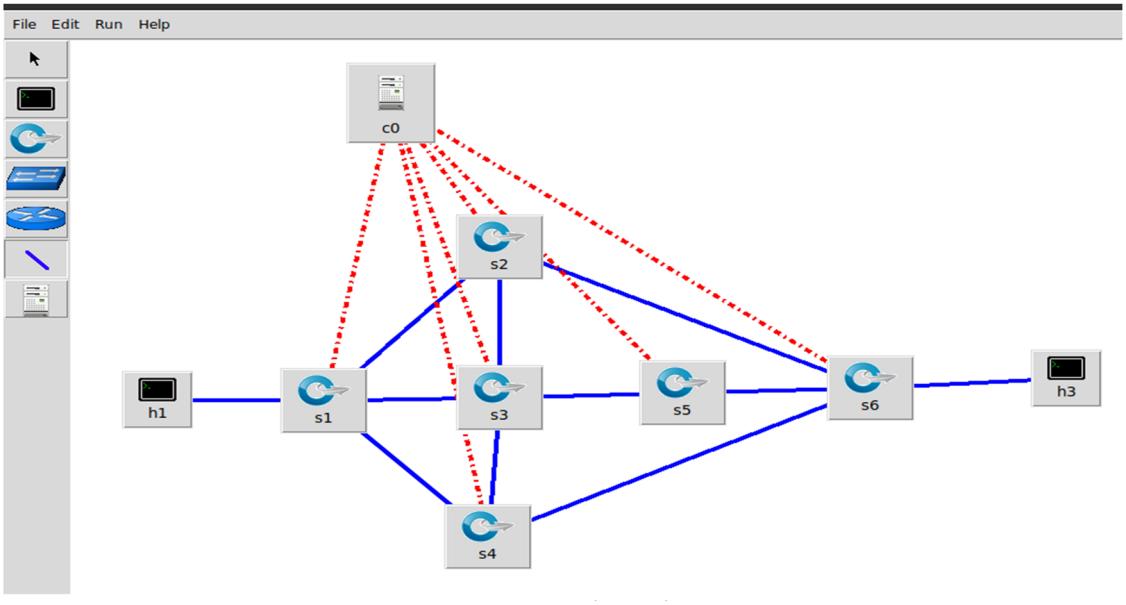


Figure 7: Network topology 2.

Table 3: Path discovery and selection

Path discovery	Hops	Path selection
s1–s2–s6	3	√
s1–s3–s5–s6	4	×
s1–s4–s6	3	√
s1–s2–s3–s5–s6	5	×
s1–s3–s2–s6	4	×
s1–s4–s3–s5–s6	5	×
s1–s4–s3–s2–s6	5	×
s1–s2–s3–s4–s6	5	×

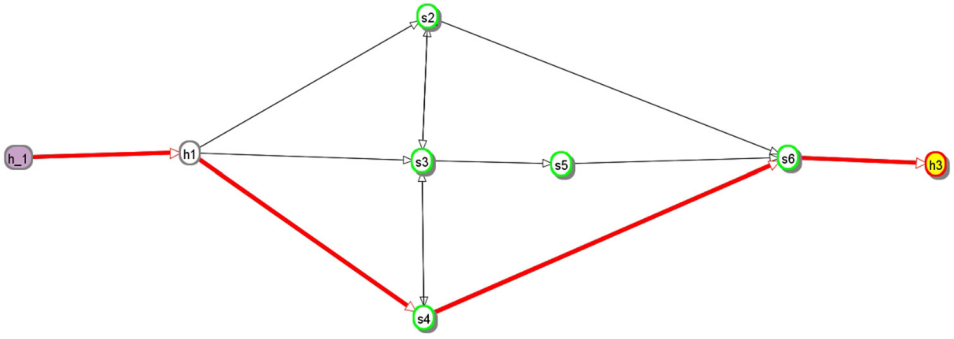


Figure 8: Path selection.

Figure 8 depicts the firefly search algorithm's selection route. Figure 7 was utilized in the scenario to show the optimum path choice.

As we can see in Figures 3 and 7, two different topologies were created. These topologies used to present and apply our proposed method to select short path between source and destination. Figure 6 depicts the ping request from host 1 to host 3 and the successful completion with the port, MAC address, and short path as indicated in Figure 4. Table 2 shows that different paths were discovered but only the short path is selected as best path from source to destination. Our proposed method achieved good result (short path). The proposed method is applied only on Mininet server and used a SDN environment with Ryu.

6 Conclusion and future works

The firefly search method has been implemented using Mininet and Ryu in a SDN environment to illustrate dynamic programmability utilizing controllers in this study. The SDN field is relatively new, yet it is rapidly expanding. Significant research concerns must be addressed, such as security and load balancing. If a company wants a specific network behavior, it can create or install an application to meet its needs. This application might be a typical networking function like traffic engineering, policy routing, firewalling, or security. SDN can improve the efficiency of deploying and managing network applications and services. Load balancing and firewalling can be imitated in the future, depending on our campus needs. Because SDN allows developers to create applications depending on particular campus requirements, such as during an online examination at a university, priority and higher bandwidth may be given to Html sites during that hour. Load balancing can also be done based on the link's cost and the number of controllers in use. We plan to extend this research with other optimization algorithms (PSO, cuckoo search, etc.) and compare between them.

Conflict of interest: The authors declare no conflict of interest.

References

[1] Yin H, Jiang Y, Lin C, Luo Y, Liu Y. Big data: transforming the design philosophy of future internet. IEEE Netw. 2014; 28(4):14–9.

[2] Jain A, Pasquale J. Internet distance prediction using node-pair geography. 2012 IEEE 11th International Symposium on Network Computing and Applications. IEEE; 2012. p. 71–8.

[3] Rouskas GN, Baldine I, Calvert K, Dutta R, Griffioen J, Nagurney A, et al. Choicenet: Network innovation through choice. 2013 17th International Conference on Optical Networking Design and Modeling (ONDM). IEEE; 2013. p. 1–6.

[4] Rothenberg CE, Nascimento MR, Salvador MR, Corrêa CN, Cunha de Lucena S, Raszuk R. Revisiting routing control platforms with the eyes and muscles of software-defined networking. Proceedings of the first workshop on Hot topics in software defined networks. 2012. p. 13–8.

[5] Gupta A, Vanbever L, Shahbaz M, Donovan SP, Schlinker B, Feamster N, et al. Sdx: A software defined internet exchange. ACM SIGCOMM Computer Commun Rev. 2014;44(4):551–62.

[6] Fortz B, Thorup M. Internet traffic engineering by optimizing OSPF weights. Proceedings IEEE INFOCOM 2000. conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (*Cat. No. 00CH37064*). Vol. 2. IEEE; 2000. p. 519–28.

[7] Ericsson M, Resende MGC, Pardalos PM. A genetic algorithm for the weight setting problem in OSPF routing. J Comb Optim. 2002;6(3):299–333.

[8] Srivastava S, Agrawal G, Pioro M, Medhi D. Determining link weight system under various objectives for OSPF networks using a Lagrangian relaxation-based approach. IEEE Trans Netw Serv Manag. 2005;2(1):9–18.

[9] Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, et al. B4: Experience with a globally-deployed software defined WAN. ACM SIGCOMM Computer Commun Rev. 2013;43(4):3–14.

[10] Hong CY, Kandula S, Mahajan R, Zhang M, Gill V, Nanduri M, Wattenhofer R. Achieving high utilization with software-driven WAN. Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM. Vol. 5, 2013. p. 15–26.

[11] Agarwal S, Kodialam M, Lakshman T. Traffic engineering in software defined networks. 2013 Proceedings IEEE INFOCOM. IEEE; 2013. p. 2211–9.

[12] Hu Y, Wang W, Gong X, Que X, Ma Y, Cheng S. Maximizing network utilization in hybrid software-defined networks. 2015 IEEE Global Communications Conference (GLOBECOM). IEEE; 2015. p. 1–6.

[13] Wang W, He W, Su J. Boosting the benefits of hybrid SDN. 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE; 2017. p. 2165–70.

[14] Guo Y, Wang Z, Yin X, Shi X, Wu J. Traffic engineering in SDN/OSPF hybrid network. 2014 IEEE 22nd International Conference on Network Protocols. IEEE; 2014. p. 563–8.

[15] Hong DK, Ma Y, Banerjee S, Mao ZM. Incremental deployment of SDN in hybrid enterprise and ISP networks. Proceedings of the Symposium on SDN Research; 2016. p. 1–7.

[16] Jin C, Lumezanu C, Xu Q, Mekky H, Zhang Z-L, Jiang G. "Magneto: Unified fine-grained path control in legacy and openflow hybrid networks. Proceedings of the Symposium on SDN Research; 2017. p. 75–87.

[17] Chu C-Y, Xi K, Luo M, Chao HJ. "Congestion-aware single link failure recovery in hybrid SDN networks. 2015 IEEE Conference on *Computer Communications* (*INFOCOM*). IEEE; 2015. p. 1086–94.

[18] Caria M, Das T, Jukan A, Hoffmann . Divide and conquer: Partitioning OSPF networks with SDN. 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). IEEE; 2015. p. 467–74.

[19] He J, Song W. Achieving near-optimal traffic engineering in hybrid software defined networks. 2015 IFIP Networking Conference (IFIP Networking). IEEE; 2015. p. 1–9.

[20] Xu H, Fan J, Wu J, Qiao C, Huang L. Joint deployment and routing in hybrid SDNs. 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS). IEEE; 2017. p. 1–10.

[21] Xu H, Huang H, Chen S, Zhao G. Scalable software-defined networking through hybrid switching. IEEE INFOCOM 2017-IEEE Conference on Computer Communications. IEEE; 2017. p. 1–9.

[22] Vissicchio S, Vanbever L, Rexford J. Sweet little lies: Fake topologies for flexible routing. Proceedings of the 13th ACM Workshop on Hot Topics in Networks; 2014. p. 1–7.

[23] Ghafori S, Gharehchopogh FS. Advances in spotted hyena optimizer: a comprehensive survey. Arch Computat Methods Eng. 2021;29:1–22.

[24] Lantz B, Heller B, McKeown N. A network in a laptop: rapid prototyping for software-defined networks. Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks; 2010. p. 1–6.

[25] Ozdemir S, Xiao Y. Secure data aggregation in WSNs: A comprehensive overview. Computer Networks. 2009;53(12):2022–37.

[26] Mohaisen A, Nyang D-H, AbuHmed. T. Two-level key pool design-based random key pre-distribution in wireless sensor networks. KSII Trans Internet Inf Syst (TIIS). 2008;2:222–38.
[27] Mohaisen A, AbuHmed T, Zhu T, Mohaisen M. Collaboration in social network-based information dissemination. In 2012 IEEE International Conference on Communications (ICC). IEEE; 2012. p. 2103–7.
[28] Yang X-S. Nature-inspired optimization algorithms. Challenges and open problems. J Comput Sci. 2020;46:101104.
[29] Goldanloo MJ, Gharehchopogh FS. A hybrid OBL-based firefly algorithm with symbiotic organisms search algorithm for solving continuous optimization problems. J Supercomputing. 2022;78:3998–4031.
[30] Abedi M, Gharehchopogh FS. An improved opposition based learning firefly algorithm with dragonfly algorithm for solving continuous optimization problems. Intell Data Anal. 2020;24:309–38.