2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA) – IRAQ (9-10) May

Routing Algorithm Optimization for Software Defined Network WAN

Ameer Mosa Al-Sadi
University of Northampton
Department of Computing and Immersive Technologies
Northampton, United Kingdom
Ameer.alsadi@northampton.ac.uk

Ali Al-Sherbaz
University of Northampton
Department of Computing and Immersive Technologies
Northampton, United Kingdom
Ali.Al-Sherbaz@northampton.ac.uk

James Xue
University of Northampton
Department of Computing and Immersive Technologies
Northampton, United Kingdom
James.Xue@northampton.ac.uk

Scott Turner
University of Northampton
Department of Computing and Immersive Technologies
Northampton, United Kingdom
Scott.Turner@northampton.ac.uk

Abstract- Software Defined Network (SDN**) provides a new fine-grained interface enables the routing algorithm to have an a global view of the network throughputs, connectivity and flows at the data-path. This paper aims to provide a novel approach for dynamic routing algorithm for Software Defined Network in Wide Area Network (**SDN-WAN**); based on using a modified shortest-widest path algorithm with a fine-grained statistical method from the OpenFlow interface, called Shortest-Feasible OpenFlow Path (SFOP). This algorithm is designed to identify the optimal route from source to destination, providing efficient utilization of the SDN-WAN resources. It achieves this aim by considering both the flow requirements and the current state of the network. SFOP computes the optimal path which provides the feasible bandwidth with the lowest hop count (delay). That will present better stability in SDN communication, QoS, and usage of available resources. Moreover, this algorithm will be the base for an SDN controller because it extracts the widest available bandwidth from source to destination for a single path. It enables the controller to decide whether it is enough to use this simple algorithm only, or if a more complicated algorithm that provides larger bandwidth such as multiple-path algorithms is needed. Finally, a testbed has been implemented using MATLAB Simulator, Pox controller, and Mininet emulator will be discussed. The latency comparison of SFOP algorithm with three other algorithm's latencies shows that this algorithm finds better latency for an optimal path. Evidence will be shown that demonstrates that SFOP has good stability in dynamic changes of SDN-WAN.**

Keywords—SDN; WAN; Matlab; Mininet; OpenFlow; Pox.

I. INTRODUCTION

Traditional networks have very distributed control mechanisms. Specific routing protocols and algorithms need to be executed on every node. They find the best route from the source to the destination and keep usage balance of network resources [1]. The rapid growth of the Internet makes the network managing and monitoring a difficult task. Therefore, the Traffic Engineering (TE) is used to optimize controlling and monitoring the flow in the network. TE helps to satisfy a specific level of Quality of Service (QoS) and providing efficient resources utilization of network [2].

However, TE faces two challenges: 1) it increases the communication; and 2) the computation overhead.

Regarding the first challenge, the flow-base techniques cause extra communication to reserve the path resources. Moreover, the routing algorithms related to TE techniques make a heavy measurement to learn the available resources and their utilization. Consequently, they require a greater level of communication to build and update their database. All of which raise the amount of the communication in the network [3].

The second challenge, the complicated algorithms related to TE require appreciable computation complexity both in terms of time and space. Therefore, this level of consumption of computation power within network nodes makes it difficult to provide these requirements in all nodes of the network [2].

Fortunately, a novel paradigm of networking called Software-defined Network (SDN) has diminished both of these challenges.

The most important achievement of SDN is that it significantly reduces the communication overhead needed to have a global view of network resources utilization. Also, SDN central control reduces the number of authorities, which requests this information [3]. SDN reduce communication overhead through its use of a novel interface between network nodes and a controller such as OpenFlow protocol. This interface contains counters, which indicate the state of network throughputs, connectivity, and flows [3][4]. Therefore, this paper suggests using OpenFlow statistics to compute the flow path from source to destination.

A further point is that SDN increases the flexibility in the use of routing algorithms by changing the way, the location, and the frequency of the routing algorithms are implemented [5]. In logically centralized control, reserving a path from the source to the destination for each flow can be achieved without flow-based techniques; however, the flow-based techniques in SDN are available if required. In addition, the routing algorithms now only run in the controller and for one time per flow in a normal scenario. As a result, SDN can provide a more precise path with less delay.

In this paper, we present the use of the statistics of the OpenFlow protocol with the developed ***SFOP*** algorithm, which is better than simple algorithms such as shortest, widest, or shortest-widest path for the following reasons:

1. The ***SFOP*** algorithm will find the optimal path:
   a) Has a lower latency by computing the shortest path of feasible bandwidth.
   b) It will not overuse the available bandwidth like widest and shortest-widest path routing algorithms [6]. Because it will use the feasible bandwidth as a base to search the shortest path.

   It does those without raising the time and space of the computation complexity to the limit, which could produce an unacceptable delay in performance of SDN controller.
2. This algorithm will be the basis of the controller, which separate implementing a simple routing algorithm for the normal path, from the implementation of a more complicated one for an intractable path. It supplies the controller with a maximum bandwidth of the widest path. This data facilitate the controller to decide whether it needs to implement more complicated routing algorithm such multi-flow algorithm [7][8].
3. The fine-grained statistic of OpenFlow will reduce the costs of measurement and computation needed to build algorithm database [3].
4. The good convergence of OpenFlow statistic will guide the routing algorithm to use real available nodes capacity and links bandwidth, which optimize resources utilization [3].

In this paper, we use an SDN emulator namely Mininet [9] and a Pox controller, which is an open source development platform for Python-based SDN control applications [10], to emulate the paths of the dynamic routing algorithm, while, a path is computed through MATLAB Simulator tools.

The rest of this paper is structured as the following: section II briefly reviews the related work; section III explains the statistics of OpenFlow interface; the SOFP algorithm is demonstrated in section IV; section V conducts algorithm complexity analysis; section VI introduces the testbed; the emulation results are discussed in VII. Finally, conclusions of the work are shown in section VIII.

II. RELATED WORKS

History of routing algorithms for the traditional network is evolved from simple ones such as shortest and the widest path to the very complicated algorithms of constraint QoS [7]. Many works evaluate the performance of routing algorithm and classified them according to their complexity and efficiency for a specific type of traffic [6][11][12]. The work in [6] concluded that shortest path algorithm works fine for light traffic loads and the widest-shortest path algorithm is convenient for heavy loads. Widest path algorithm performs badly for both heavy and light loads. While, the routing algorithm of shortest-widest path causes a low throughput for best-effort-traffic.

After that, a more complicated routing algorithm was developed to satisfy specific QoS requirements. Some authors provided intensive review for QoS routing algorithm [12][13]. Multi-Constrained Multipath (MCMP) routing algorithm was designed using a combination of multi-routing algorithms and redefined matrices [11]. On team [13] developed an algorithm to find bandwidth and delay constrained path. It can be seen that routing algorithms have attracted a great deal of research interest in the traditional network.

The emergence of SDN requires reinvestigation of the routing algorithm due to the new features of a novel paradigm.

On one hand, several works implemented using a simple algorithm such as shortest and widest routing algorithm to optimize routing in SDN. For example, [14] used the widest path to find a path between switch and server in the data center. While in [15] the widest or shortest path was used after application server notifying the controller which one is more suitable for the current flow. Finally, one of the recent work in this area [5], they replaced the shortest path with widest path algorithm in a Floodlight controller. As noted, this work did not develop many new aspects for a simple polynomial routing algorithm to exploit new features of SDN or to satisfy its new requirements. They mostly implement and evaluate the traditional algorithms in SDN network. Therefore, this paper aims to fill this gap.

On the other hand, the majority of the routing algorithm optimization in SDN focused on complicated routing algorithms, which are destined to serve many constrain QoS specifications. For instance, [1][16] used very complicated algorithms to optimize QoS for multimedia and video streaming. Similar work in [17], a multi-constrained shortest path to enhance video streaming was constructed. Subsequently, [18] worked to get better QoS with minimal controller load at the same time. After that, [8] utilized the advanced Boolean satisfiability (SAT) techniques to compute the constrained shortest path. Some researchers [19] have employed Bayes' probability theorem to find the feasible link of satisfied QoS constraint. Eventually, K maximally disjoint path algorithm was presented in [20]. This work also aimed to provide a buffer, which prevents implementing these complicated algorithms until the flow requirements enforce the controller to implement them.

This work develops SFOP algorithm, which provides a path of lower latency and feasible bandwidth. At the same time, it keeps acceptable complexity according to the SDN controller capabilities. In addition, it focuses on replacing the informative old protocol, and the complex computation requirements to update the resources utilization; by a fine-grained OpenFlow interface, which is one of the main keys to the success of SDN.

Finally, it is clear that the shortest-widest path algorithm consuming the network bandwidth negatively, as mentioned in the first paragraph. Therefore, it is modified to search the shortest path of feasible bandwidth for the flow.

III. STATISTICS OF OPENFLOW INTERFACE

This work suggests using the fine-grained statistics of OpenFlow interfaces to compute the optimal path by routing algorithms. Especially, the information required for routing algorithms extracted by efficient open-source software's, such as Open Traffic matrix (Open TM) [21] and Open Network Monitoring (OpenNetMon) using OpenFlow interface [3].

This section briefly shows OpenFlow protocol and the related routing algorithms requirements. OpenFlow protocol is

designed to have a global view of all network connectivity and throughput for dataflow layer. When any change in network connectivity happen, OpenFlow switches send a ***port-state message*** to the controller to update this change [22]**.** In addition, OpenFlow switches have flow counters ***per-port***, ***per-flow***, and ***per-queue***. These counters maintain the network state in controller database using two types of messages. The more frequent one is the ***flow-removed*** message when the flow time up. The second message is ***Read-State***, which is sent by the controller to collect switches statistics. Both messages contain the ***flow-duration*** and its ***byte count*** at each switch, which enables to identify network throughput for each link [22][23].

The developed routing algorithm needs to learn the network throughput as an incident matrix of bandwidth (***IncBW***) [3][21]. In addition, this software should supply the algorithm by the feasible bandwidth per flow type (***FBW***) based on a monitoring and the QoS requirements [4][3].

The algorithm also uses the OpenFlow to identify the type of flow based on different parameters in OpenFlow header, which are source port (***TCP/UDP-src-port***), destination port (***TCP/UDP-src-port***) and type of service (***IP-TOS***) [22].

IV. THE SFOP ALGORITHM

It is based on a combination of shortest widest path and the restricted shortest path. The algorithm is executed as a management application inside the SDN controller. The algorithm has two input parameters, which are the incident matrix of residual bandwidth (**IncBW**) in the current state of the network and the feasible bandwidth (**FBW**) for the current flow. Both are learned from the open source software presented in [3][4], which use the fine-grained OpenFlow statistic as explained in section III.

First, the algorithm starts by computing the widest path and widest bandwidth. Second, it is modified to compare the FBW of OpenFlow statistics with the found widest bandwidth. Then, it uses the FBW if it is less or equal to the widest bandwidth. Otherwise, it uses the widest bandwidth itself. Third, it finds the shortest path of the chosen bandwidth. The second and third parts (the modified parts) are the contributions of this paper. Unlike the algorithm of the shortest widest path, that finds the shortest path of widest bandwidth. The algorithm of SFOP finds the shortest path of feasible bandwidth.

The algorithm computes the optimal path, which provides feasible bandwidth and the minimum hops count (minimum delay). In the meantime, the algorithm also optimizes the utilization of links bandwidth, because the algorithm will always use the path, which has the feasible bandwidth in the current state of SDN-WAN and leaves the links, which have the widest bandwidth for best-effort-traffic.

Let's start to formulate the algorithm for a more detailed demonstration. The SDN-WAN is represented by undirected finite graph $\boldsymbol{G}\ (\boldsymbol{N}, \boldsymbol{L})$, where the $\boldsymbol{N}$ is the network nodes and $\boldsymbol{L}$ is the links between them. $|N| = n$ and $|L| = l$ .The path $\boldsymbol{P}$ is the optimal path of sequence nodes $\boldsymbol{P} = (\boldsymbol{In} = \boldsymbol{n1}; \boldsymbol{n2}, \dots, \boldsymbol{nk} = \boldsymbol{Eg})$, where $\boldsymbol{In}$ is Ingress node, $\boldsymbol{Eg}$ is Egress node, $\boldsymbol{k}$ is the length of the path and $(n_i; n_{i+1}) \in L : \forall i = (1, \dots, k-1)$. A link $\boldsymbol{l}$ with origin node $\boldsymbol{n}$ and destination node $\boldsymbol{m}$ is denoted by $(\boldsymbol{n},\boldsymbol{m})$. Respectively, with each link $\boldsymbol{l} = (\boldsymbol{n}, \boldsymbol{m}), \boldsymbol{n}, \boldsymbol{m} \in \boldsymbol{N}$ there is an associated bandwidth $\boldsymbol{BW_l \geq 0}$ and a delay $\boldsymbol{\delta_l \geq 0}$.

The main steps of SFOP algorithm:

1. Find the less utilize links connect every neighbor of nodes (create Adjacency matrix **AdjBW** from incident matrix **IncBW**. See pseudocode in figure1.
2. Compute the widest path using modified Dijkstra's algorithm. See figure 2.
   The widest path is defined as shown in equation **1**:
   $$\boldsymbol{Bandwidth(widest_P)} = \boldsymbol{Max}_{l_i=(l_1,l_2,l_3\dots l_l:\ l_i \in Adl\ and\ I \leq ADC)} [\mathbf{min}(\boldsymbol{BW_{li}}, \boldsymbol{D_i})] \dots (\mathbf{1})$$
   Where:
   a) *D = Distance matrix, which save the maximum bandwidth of path from the ingress node until this node.*
   b) *Adl = the group of links connected to adjacent nodes.*
   c) *Adc = the count of these links.*
3. Check bandwidth; algorithm uses the feasible bandwidth (FBW) if it is less than the widest bandwidth ($widest(BW_P)$). Otherwise the widest bandwidth ($widest(BW_P)$) is used. See figure 3.
4. Remove all the links have bandwidth less than the last specified bandwidth ($widest(BW_P)$ or FBW) . See figure 3, step 31.
5. Compute the shortest path of last specified bandwidth, using another modified Dijkstra's algorithm. See figure 4.
   The cost of shortest path is defined in equation **2**:
   $$\boldsymbol{Cost(shortest_P)} = \sum_{k=0}^{N} \underset{l_i=(l_1,l_2,l_3\dots l_l:\ l_i \in Adl\ and\ I \leq Adc)}{\mathbf{Min}} [\boldsymbol{BW_{li}} + \boldsymbol{D_i}] \dots (\mathbf{2})$$
   Where:
   a) *D = Distance matrix, which saves the accumulated bandwidth of path from the ingress node until this node.*
   b) *Adl = the group of links connected to adjacent nodes*
   c) *Adc = the count of these links.*

Initialize 1# create **Adjacency matrix of bandwidth (AdjBW)** by extracting it from Incident matrix of the graph (**IncBW**).
(G: GRAPH, NODE= N, Links=L, Incident matrix of bandwidth=**IncBW**, Adjacency matrix of bandwidth = **AdjBW**)
1. Input: incident matrix of bandwidth
2. **For** $all\ n \in N$ **do** . */* for all nodes in G */*
3. chose the link of higher bandwidth;
   / to find the less utilize link if there are more than one link */*
4. **Endfor**

Fig. 1. Pseudo code for creating adjacency matrix of bandwidth.

Algorithm1# compute the widest path##########################.
(Ingress node=**In**, Egress node=**Eg,** Visited nodes vector=***V***, Unvisited node vector=***UV***, Distance matrix=**D**, Current Visit Node **=CVN,** path=**P,** Path bottleneck bandwidth= ***widestBW***).
Compute distance matrix
5. **Initialize**: $V = \emptyset$; */* initialize visited node vector to be empty*/*
6. **Initialize**: $UV = all\ N$; */* initialize Unvisited node vector to have all N*/*
7. **Initialize**: k=0; */* initialize counter to zero*/*
8. **Initialize**: $\boldsymbol{CVN = 0}$; */* variable to hold the Current Visit Node (CVN) */*
9. **While** $UV \neq \emptyset$ **do** */* for all nodes in UV */*
10. **k=k+1**; */* increment counter*/*
11. $\boldsymbol{CVN = maxdistance}(n)$
    */*Select node n has maximum distance D to be current visited node*/*

```
12.   Insert CVN in V
13.   Remove CVN from UV
14.   For all neighbours [CVN] do
15.       D = [ min(AdjBW_lk, D_k−1)] /* update the bottleneck
          bandwidth in D for all neighbors of CVN whose yet unvisited */
16.   Endfor
17. EndWhile
# ## Compute widest path and its bandwidth from distance matrix.
18 Initialize: P= Eg; /* initialize path vector (P) to equal Egress node*/
19. Initialize: temp-node=Eg; /* initialize variable to hold last node in path*/
20. Initialize:widestBW = ∞;
    /* initialize variable to hold the bottleneck bandwidth of widest path*/
21. While temp − node ≠ In do
22.     temp − node = last node of P
23.     Insert n( max ( D of all neighbours_temp−node)) to P
        /* Update path vector*/
24.     widestBW = min(previous widestBW, max D)
25.  Endwhile
```

Fig. 2. Pseudo code of algorithm 1 (computing the widest path).

```
Initialize2# Find and Use Best Available Bandwidth ########.
( Feasible bandwidth of flow =FBW, Best Available bandwidth = BABW).
# Best Available bandwidth ( BABW ) equal to the feasible or widest bandwidth.
26. Input: FBW;  /*input the Feasible bandwidth (FBW) for this flow */
27. Initialize: BABW = 0;
    /*Variable hold the value of Best Available bandwidth (BABW) */
28. If FBW ≥ widestBW then BABW = FBW
    /*if the feasible bandwidth is available then use it*/
29. Else BABW = widestBW
30. Endif
#Remove any bandwidth lower than BABW from adjacency matrix of bandwidth
(AdjBW ).
31.  For all n ∈ N do  /* for all nodes in G */
32.    If BW_n < BABW then BW_n = 0; /* set unwanted bandwidth to zero*/
33. Endfor
```

Fig. 3. Pseudo code for finding and using the best available bandwidth.

```
Algorithm 2#compute the shortest path##########################.
#prepare adjacency matrix of bandwidth (AdjBW ).
34. For all n ∈ N do /* for all nodes in G */
35.    If AdjBW_n ≤ 0 then AdjBW_n = ∞;
       /* set zero bandwidth to infinity*/
36. Endfor
# updates node distance and path.
37. Initialize: V = ∅; /* initialize visited node vector to be empty */
38. Initialize: UV = all N;
    /* initialize unvisited node vector to have all nodes N*/
39. Initialize: k=0 /* initialize counter to zero*/
40. Initialize: CVN = 0
    /* it is a temporary variable to hold the Current Visit Node (CVN) */
41. While UV ≠ ∅ do /* for all nodes in UV */
42.    k=k+1; /* increment counter*/
43.    CVN = mindistance(n); /*Select n with minimum distance D */
44.    Insert CVN in V;
45.    Remove CVN from UV
46.    For all neighbours [chosen] do
47.        D = [ AdjBW_lk + D_k−1] /* update the bottleneck bandwidth
           in distance matrix D for all neighbors of CVN whose yet unvisited */
48.        P = n( min ( D of all neighbours_temp−node))
           /*Update shortest path*/
49.    Endfor
50. EndWhile
```

Fig. 4. Pseudo code of Algorithm 2 (compute the shortest path).

V. Computational Complexity Analysis

As have been shown in the previous section the SFOP algorithm is composed of two main parts. Both parts are based on modified Dijkstra's algorithm [24]. Algorithm 1, shown in figure 2, is responsible for computing the widest path. The worst-case complexity of Algorithm 1 is $O(NlogN + L)$ [25], where N is network nodes, and L is network links. Similarly, Algorithm 2's computational complexity is $O(NlogN + L)$[24], where N is the number of network nodes, and L is network links. Therefore, the worst-case computational complexity for all algorithm is $O(NlogN + L)$.

From a simple comparison of computational complexity, between this algorithm and other polynomial routing algorithms, has been presented in [26]. It has been observed that most of the polynomial algorithms used to find the optimal path have a similar time complexity of our algorithm. However, SFOP algorithm still reduces time complexity because in logically centralize control of SDN it is executed fewer times in comparison to distributed control of the traditional network.

VI. The SFOP Emulation Environment

This section introduces the hardware and software tools. A computer with a corei7 of the 2.4GH processor and 16GB memory is used to implement the emulation. The Matlab programming language is used to write the management routing algorithms. This program reads the incident matrix of bandwidth (IncBW) and feasible bandwidth of current flow (FBW). After that, it computes the optimal path (P) using the SFOP algorithm.

Next, a Python application program is developed for the Pox controller to read the optimal path and install the require rules to OpenFlow switches. In addition, it updates the bandwidth matrix of SDN-WAN. Pox controller is used due to it is easy to program and has no faults [27].

Another, Python program is written to create the SDN-WAN topology. Finally, Mininet emulator is used to executing the Pox controller and emulates the network of OpenFlow virtual switches (OVS).

VII. The Implementation and Results

A virtual network is generated to emulate SDN-WAN. This network is designed to have as similar characteristic to the real WAN topology as much as possible, as shown figure 5.

The topology has one Pox controller and sixteen OVS switches, which is the maximum port capacity for a Pox controller. The bandwidth backbone links are 1 Gbit/s while the forked links have 100 Mbit/s and 10 Mbit/s bandwidth.

A Pox controller is placed in the position of node eleven because it locates in center of SDN-WAN and lays on its backbone infrastructure.

Two different tests are implemented in this algorithm. The first one is to compare the path latency of SFOP algorithm with the path latency of shortest, widest and shortest-widest

algorithms to specify which algorithm has the best performance. The second comparison is for path latency of SFOP algorithm in multiple states of the network to evaluate the algorithm performance with dynamic changes.

Both tests apply on three sizes of packets (1 KB, 10 KB, and 64 KB) to evaluate the performance of the routing algorithm with different loads. Moreover, both of them use single ingress node and Egress node.

A) Test one: it shows the different paths of the routing algorithms as shown in figure 5.

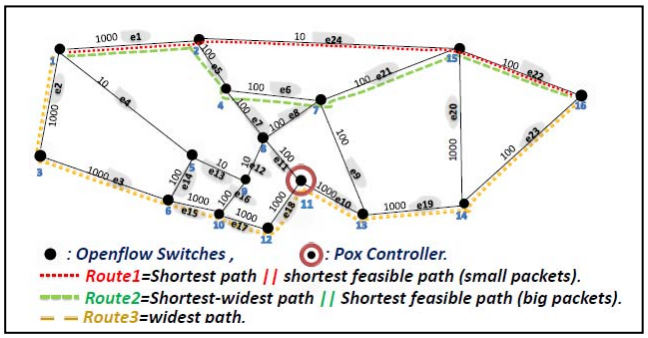

Fig. 5. Different routing algorithms in SDN-WAN.
Note: 10,100 and 100 represent the link bandwidth in Mbit/s.

Route one shows the shortest path. It also represents SFOP when small packets are sent (feasible bandwidth required is less than 10 Mbits/s). Route two displays the shortest path. In addition, it presents SFOP when packet size is larger than 10 KB (a feasible bandwidth larger than 10 Mbits/s is specified). Finally, route three expresses the widest path. Consequently, the emulation generates the following results. See figure 6.

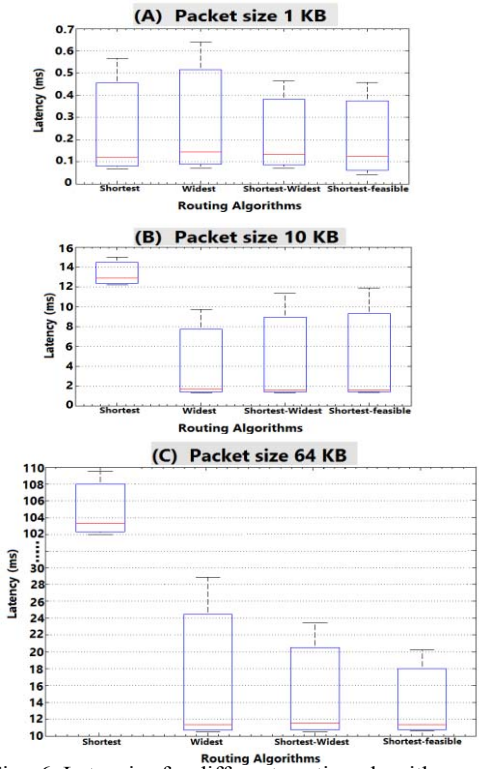

Fig. 6. Latencies for different routing algorithms.

These results show that the shortest path and SFOP algorithms have the best mean of latency for small packets. While, for big packets the shortest path algorithm, is very bad, and the other three have similar results. Which mean the shortest-feasible bandwidth do better than others in all loads. See the comparison summary in table 1.

TABLE 1. Latency performance of different routing algorithms.

No.	Packet size	Shortest (ms)	Widest (ms)	shortest-widest (ms)	shortest-feasible (ms)
1	1 KB	Best (0.108)	Good (0.147)	Better (0.125)	Best (0.111)
2	10 KB	Bad (12.894)	Best (0.145)	Best (1.607)	Best (1.581)
3	64 KB	Bad (03.345)	Best (11.342)	Best (11.517)	Best (11.357)
Notes:				Consume residual Bandwidth	Not consume residual Bandwidth

B) Test two: it shows different paths of SFOP algorithm when the network links have a different residual bandwidth (IncBW), and the feasible bandwidth (FBW) is fixed to be larger than 100 Mbit/s. See figure 7.

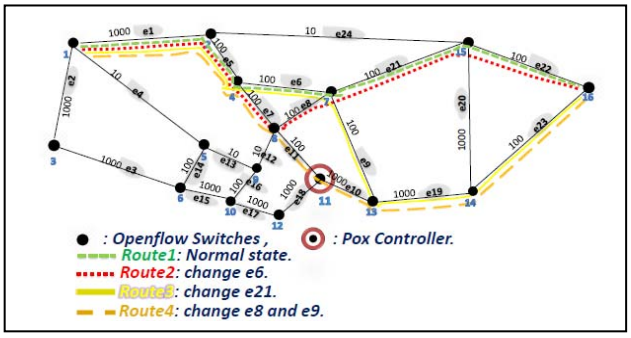

Fig. 7. Several paths of SFOP algorithm in dynamic bandwidths changes.

Route one shows the path in a normal state (full bandwidths). Route two displays the path if the bandwidth of link **e6** down from 100 Mbit/s to 10 Mbit/s. The third route presents the path in case link ***e*21** goes from 100 Mbit/s to 10 Mbit/s. Finally, route four, represent the path when links ***e*8** and ***e*9** have residual bandwidths 10 Mbit/s instead of 100 Mbit/s in initial state.

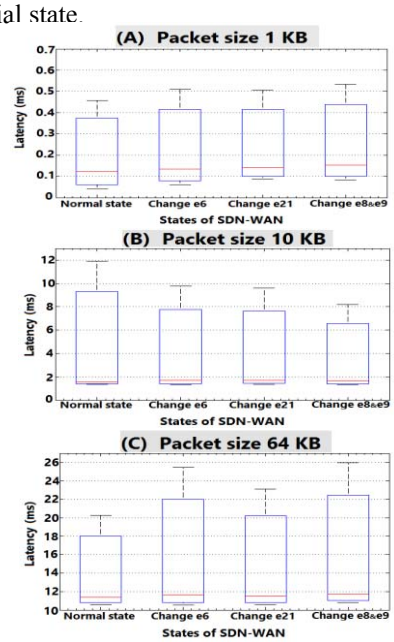

Fig. 8. Latency of dynamic routing in different states of SDN-WAN.

As a result, the algorithm shows similar latency for all cases if an alternative path of similar bandwidth is available. See figure 8. In summary, the algorithm provides good stability for flow path in SDN-WAN.

Bearing in mind that the test is done using ping command with different bandwidths to examine the algorithm latency, which is the focus of this paper. Heavier tests will be considered in future to qualify the faults and losses packets.

VIII. CONCLUSION

In this paper, we proposed a routing algorithm called SFOP to find the optimal path and to enhance the network resources utilization. In SDN-WAN, the algorithm emulation in Mininent, shows that it provides better latency than other comparable polynomial routing algorithms (shortest, widest and shortest-widest path). It also shows it has a good stability with dynamic changes of the network in most cases. Moreover, OFSP provides valuable data for the controller to build its decision to apply the further complicated algorithm as demonstrated in section IV. To sum up, the SFOP algorithm works better than others in term of latency and resources utilization of SDN-WAN without increment of the computation complexity. The computation complexity and communication overhead are significantly reduced due to use the fine grand statistics of OpenFlow interface, which is the key success of SDN and this algorithm as well.

The future works aim to implement this algorithm for the larger topology of SDN-WAN, which constructs from multiple controllers. We also aim to test this algorithm on other types of the controller such as Floodlight and Opendaylight controllers [27]. At the end, we work toward implementing the same approach of using the fine-grand static interface of OpenFlow to develop routing algorithms to compute multipath and backup path.

ACKNOWLEDGMENT

This work was supported in part by Iraqi Ministry of Higher Education and Scientific Research - scholarship no.21573 for the first author.

REFERENCES

[1] H. E. Egilmez, B. Gorkemli, A. M. Tekalp, and S. Civanlar, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *2011 18th IEEE International Conference on Image Processing*, 2011, pp. 2241–2244.

[2] P. Trimintzios, G. Pavlou, and I. Andrikopoulos, "Providing Traffic Engineering Capabilities in IP Networks Using Logical Paths," *, Eighth IFIP Work. Perform. Model. Eval. ATM IP Networks (IFPM ATM IP 2000), Ilk. UK July*, 2000.

[3] V. Adrichem, N. L. M., C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–8.

[4] H. Jin, D. Pan, J. Liu, and N. Pissinou, "OpenFlow-Based Flow-Level Bandwidth Provisioning for CICQ Switches," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1799–1812, Sep. 2013.

[5] K. G. Yalda, D. J. Hamad, and I. T. Okumus, "Design and Implementation of an Intra-domain routing module for an SDN controller for Traffic Engineering in SDN environment," in *2015 International Conference on Advances in Software, Control and* Mechanical *Engineering (ICSCME-2015) Sept. 7-8, 2015 Antalya (Turkey)*, 2015, p. 93.

[6] Q. Ma and P. Steenkiste, "Routing Traffic with Quality-of-Service Guarantees in Integrated Services Networks," *Proc. NOSSDAV '98,* Cambridge*, UK,* Jul. 1998.

[7] G. R. Márton Zubor , Attila Kőrösi, András Gulyás, "On the Computational Complexity of Policy Routing," vol. 8846, Y. Kermarrec, Ed. Cham: Springer International Publishing, 2014.

[8] F. Aloul, B. Rawi, and M. Aboelaze, "Identifying the Shortest Path in Large Networks using Boolean Satisfiability," in *2006 3rd International Conference on Electrical and Electronics Engineering*, 2006, pp. 1–4.

[9] Mininet Team, "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet," 2015. [Online]. Available: http://mininet.org/. [Accessed: 31-Mar-2015].

[10] A. A. Shinoda, C. M. Schweitzer, and R. L. S. de Oliveira, "Simulation in an SDN network scenario using the POX Controller," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2014, pp. 1–6.

[11] P. Steenkiste and Q. Ma, "On path selection for traffic with bandwidth guarantees," in *Proceedings 1997 International Conference on Network Protocols*, pp. 191–202.

[12] S. K. SHESHADRI, "Multi-constrained node-disjoint multipath QoS routing algorithms for status dissemination networks," Doctoral dissertation,Washington State University, 2004.

[13] T. Korkmaz and M. Krunz, "Bandwidth-delay constrained path selection under inaccurate state information," *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 384–398, Jun. 2003.

[14] V. V. Rahul Cariappa Cheyanda, "Load Balanced Virtual Data *Center* Network Using SDN Approach." Department of Electrical Engineering, San Jose State University, San Jose, California, 2014.

[15] V. Ingale and S. Kakade, "Optimization of Network," Department of Electrical Engineering, San Jose State university, San Jose, *California*, 2014.

[16] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with *end*-to-end Quality of Service over Software-Defined Networks," *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*. pp. 1–8, 2012.

[17] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over *OpenFlow* Networks," *IEEE Trans. Multimed.*, vol. 15, no. 3, pp. 710–715, Apr. 2013.

[18] G. Nakibly, R. Cohen, and L. Katzir, "On the Trade-Off between Control Plane Load and Data Plane Efficiency in Software Defined Networks," Tehnical Report, Technion, Computer Science Department, 2012.

[19] A. Al-Jawad, R. Trestian, P. Shah, and O. Gemikonakli, "BaProbSDN: A probabilistic-based QoS routing mechanism for Software Defined Networks," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–5.

[20] J. O. Abe, H. A. Mantar, and A. G. Yayimli, "k -Maximally Disjoint Path Routing Algorithms for SDN," in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2015, pp. 499–508.

[21] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *11th Int. Conf.* Passive *Active Meas*, 2010, pp. 201–210.

[22] Open Network Foundation, "OpenFlow Switch Specification Version 1.0.0," 2009. [Online]. Available: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf. [Accessed: 18-Feb-2015].

[23] Open Networking Foundation, "OpenFlow Switch Specification Version 1.4.0," 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf. [Accessed: 20-Feb-2015].

[24] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Trans. Comput.*, vol. 47, no. 2, p. 263, 1998.

[25] V. Kaibel and M. Peinhardt, *On the bottleneck shortest path problem*, no. Technical Report 06–22, Konrad-Zuse-Zentrum f. Informationstechnik. Berlin, 2006.

[26] J. B. Orlin, "Max flows in O(nm) time, or better," in *Proceedings of the* 45th *annual ACM symposium on Symposium on theory of computing - STOC '13*, 2013, p. 765.

[27] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.