2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications

An Effective Path Load Balancing Mechanism Based on SDN

Jun Li, Xiangqing Chang,Yongmao Ren, Zexin Zhang, Guodong Wang

Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China

Email: lijun@cnic.cn, {changxiangqing,renyongmao,zhangzexin, wangguodong}@cstnet.cn

***Abstract*—Path load balancing is used for distributing workload across an array of paths to increase network reliability and optimize link utilization. However, it is not easy to realize the load balancing globally in traditional networks as the whole status of the network is difficult to obtain. To address this problem, we propose the Fuzzy Synthetic Evaluation Mechanism (FSEM), a path load balancing solution based on Software Defined Networking (SDN). In this mechanism, the network traffic is allocated to the paths operated by OpenFlow switches, where the flow-handling rules are installed by the central SDN controller. The paths can be dynamically adjusted with the aid of FSEM according to the global view of the network. Experimental results verify that the proposed solution can effectively balance the traffic and avoid unexpected breakdown caused by link failure. The overall network performance is also improved as well.**

Keywords—Path Load Balancing; OpenFlow; Fuzzy Synthetic Evaluation Model; Software-defined Networking

I. Introduction

The method of load balancing which can distribute workload across multiple resources, has played significant role in network. However, network load balancing is often limited to server load-balancing and link load-balancing. The server load-balancing distributes the incoming workload to an array of replicated servers to serve more clients with a minimum of latency and a maximum of throughput. A dedicated load balancer directs each client request to a particular replica. The dedicated equipment is expensive and suffers from scalability problem. And, the link aggregation is a common solution of link load balancing, which aggregates multiple physical links to a virtual logical link to transmit workload. It is usually deployed inbound or outbound of the network for avoiding a single link breakdown. Another kind of implementation of link load balancing is leveraging the MPLS Traffic Engineering (TE).But it is a complicated method that each device along the TE tunnel should be configured and each ingress device needs to compute the Constraint-based Shortest Path First(CSPF). There is no coordinator for ingress devices. MPLS TE is rarely deployed due to its complexity. All these models are useful to a certain extent, but they are relatively partial solutions to varying degrees. In these models, no entity takes a global control and the local device will be more likely to greedily select paths for their interest, the network can get stuck in locally optimal routing patterns that are globally suboptimal.

Path is composed of switch nodes and the links between switch nodes, it is defined as a way from the source node to the destination node. Path load-balancing can be used to schedule the workload from a global view in data center and backbone network. The traditional route protocols forward traffic usually according to the shortest path, which may lead to abnormal stuck induced by over-loading links. Path load-balancing technology is a good candidate to overcome this drawback. Therefore, it's very important to apply path load-balancing in the data center and backbone network that has a large number of traffic.

SDN is an developing architecture where network control is decoupled from forwarding and is directly programmable. It enables the underlying infrastructure to be abstracted for applications and network services. Network intelligence is logically centralized in software-based SDN controllers, which maintain a global view of the network[1]. SDN provides a new solution which can accelerate the research of network applications and future Internet technology. In recent years, SDN has prevailed and applied successfully in several network environments [2].

As one of the most effective southbound interfaces, OpenFlow [3] has been a key technology to realize SDN [4]. OpenFlow defines standard communication interfaces for the control layer to interact with the forwarding layer, and uses the concept of flow to identify network traffic [1]. It makes the fine-grained and flexible flow control feasible.

The SDN controller can acquire the network topology and links status to make decisions for optimally controlling network traffic. Currently, there are several SDN controllers available, for example, POX[5], NOX[6], Floodlight[7], OpenDaylight[8] etc.

SDN is able to help us implement path load-balancing from a global perspective. This paper presents an effective solution to realize path load balance based on SDN. It uses the controller to make load balance decisions with fuzzy evaluation model. Then, the controller transfer the decision to flow tables distributed in OpenFlow switches to control the final route of traffic. Numerical results show that the proposed solution can effectively plan paths with workload and increase the reliability of network.

The rest of this paper is organized as follows. Section 2 gives an overview of the related work. The FSEM model of this new solution is presented in Section 3. Section 4 describes the framework of the FSEM. Section 5 summarizes the results of the experimental measurements. Section 6 discusses some concerns. Section 7 concludes this study.

II. RELATED WORK

A few of studies have been conducted on the load balance based on the SDN technology. For example, the Plug-n-Serve [9] system uses OpenFlow to measure the state of the network and assigns the client requests for minimizing the response time from client to server. When a request arrives, the controller makes decision and installs forwarding rule that handle the rest packets of the request. However, the Plug-n-Serve system provides flexibility and reduces response time of the client requests, this solution has scalability limitations [4]. Richard Wang et al. [4] presents a more scalable solution with algorithms that compute simple wildcard rules to reduce the load on the controller and automatically adjust to changes in load-balancing policies. The solution presents a "partitioning" algorithm to determine a minimal set of wildcard rules and a "transitioning" algorithm to change the rules with the change of load-balancing policies. Marc Koerner and Odej Kao [10] develops a load balancer service which can handle the load of multiple services with the multiple OpenFlow controllers. For example, one controller handles the load for e-mail servers and the other one handles the load for web-servers. It increases efficiency by distributing workload to multiple controllers. However, it can suffer from scalability problems due to constantly change and increasing services. Hardeep Uppal and Dane Brandon [11] implemented a load-balancer architecture based on the OpenFlow technology which reduces the cost and provides the flexibility. It presents three load-balancing policies which are random-based load balancing policy to randomly select the registered servers, round robin load balancing policy to rotate the registered servers to serve the requests and load-base load balancing algorithm to select the server with the lowest load. Li-Der Chou et al. [12] presents a load balancing system based on OpenFlow with the genetic algorithm, and preconfigures the rules to the switches for flow directed in advance. The proposed system can save the cost and avoid the bottleneck of single controller.

B4 and SWAN[13,14] uses SDN in the context of inter-DC WAN. They enable an efficient inter-DC WAN by coordinating the sending rates of services and centrally configuring the network data plane to be capable of multiple transmitting using max-min method.While B4 develops custom switches and mechanisms to integrate existing routing protocols in an SDN environment, SWAN develops mechanisms for congestion-free data plane updates and for effectively using the limited forwarding table capacity of commodity switches[14]. Hui Long et al. [15] proposes a load-balance routing algorithm (LABERIO) to solve the issue of static routing path in OpenFlow enabled datacenter networks. When the link load detector exceeds the LABERIO trigger threshold, the central controller will be notified and help schedule the biggest flow on the busiest hop with the max-min remainder capacity strategy. However, all the control algorithms of the schemes we described above rely on the information of traffic demand, but traffic demand is hard to know in advance for many networks. This paper is devoted to implement path load balancing without traffic demand, it is a scheme based on the SDN with fuzzy synthetic evaluation model. And, it is a good compliment to related SDN solutions.

III. MODEL FORMULATION

In a dynamic network environment, it is difficult to absolutely define a path as the best one for a flow since there are many fuzzy factors which affect the choice. The goal and the constrains are fuzzy in nature. Path load balancing is a decision-making problem in a fuzzy network environment. To deal quantitatively with imprecision, we employ fuzzy theory to define fuzzy goals or fuzzy constrains precisely as fuzzy sets, and get fuzzy decision through an intersection of the given goals and constrains[16]. Therefore, Fuzzy Synthetic Evaluation Mechanism(FSEM) is proposed to control paths to balance workload. Since there may be many available paths for the same source node and destination node, FSEM is divided into two phases in order to guarantee efficiency. First, the Top-K paths will be selected. Second, the best path will be dynamically chosen from the Top-K paths with the aid of fuzzy synthetic evaluation method when a new flow enters the network or network status changes.

A. Top-K paths selection algorithm

It is common to have more than one path between two nodes in network, especially for those nodes which are significant or have high reliability demand. If there are too many paths, it will be inefficient to choose the best one from them in real time. So we will select Top-K paths first.

The selection procedure can be executed in two modes. One is initial mode, the other is periodic mode.

In the initial mode, there is no traffic in the network, so the Top-K paths will be the K shortest paths. The K shortest path algorithm is an extension algorithm of the shortest path algorithm, it finds K shortest paths. This algorithm is used in path load-balancing to select K alternative paths according to hop count. The K shortest algorithm based on Floyd algorithm is used in this paper. Floyd[17,18] is a dynamic programming algorithm to compute the shortest path between any two vertices in the network topology. The basic principle is as follows: Matrix map records the search procedure. It goes through each pair of vertices i and j , tries gradually to add new vertices as intermediate vertex to the original path. If the length of the new path is shorter than the original path, the original path and matrix map will update with the new path. The state transition equation is : $map[i,j] = \min\{map[i,k] + map[k,j], map[i,j]\}$.

Although the K shortest paths are initially selected according to hop count, Top-K path will be chosen with the fuzzy synthetic evaluation method（to be described in detail later） periodically when there is traffic in the network. The interval can be five minutes. In this way, the Top-K paths can adapt to the change of network traffic.

B. Fuzzy synthetic evaluation model

Multiple factors impact the evaluation of path in a vague way. Path evaluation can be conceptualized as a multiple criteria decision-making problem. Therefore, Fuzzy synthetic evaluation model for path load balancing is proposed as a multiple attribute fuzzy decision-making model.

From multiple factor perspective, the degrees of satisfiability of attributes with respect to the corresponding fuzzy set is comprehensively evaluated through membership function. Furthermore, it allows to assign a different degree of importance to each criteria. A maximizing decision is defined as a point in the space of alternatives at which the function of a fuzzy decision attains its maximum value.

The model of fuzzy synthetic evaluation for path load balancing is described as follows.

1) Determination of factor domain U for object to be evaluated.

$$U=(u_1, u_2, u_3 \cdots, u_n)$$

We consider two aspects for the routing path to be evaluated. One is the length of the path, which can be represented by hop count. The other is the traffic load along the path, which can be represented by the traffic of switches and links of the path. Since a path is composed by many switches and links, the average or the total traffic can't reflect the real status of the path, so the critical switch and link will be selected to represent the status of the path. The traffic of switch will be measured by packet count and byte count forwarded by switch. The traffic of link will be measured by the corresponding port forwarding rate. So the factor domain U for path evaluation is described as:

U = (h, p, b, r)

Where h denotes the hop count, p and b denote the transmitting packet count and byte count of the critical switch, r denotes the forwarding rate of the critical port.

2) Determination of rank domain V for each factor.

$$V = (v_1, v_2, v_3, \dots, v_n)$$

The rank domain can be composed by one or more fuzzy sets which represent the ranking of alternatives. For example, they can be (N, ZE, P), i.e, negative, zero, positive .Since we intend to choose the best path, so we define V= (P), where P represent "Positive" fuzzy set.

3) Judgment of single factor and built of fuzzy relation matrix R.

$$R = \begin{pmatrix} r_{11} & \cdots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{n1} & \cdots & r_{nm} \end{pmatrix}$$

r_{ij} suggests the degree to which the factor u_i in domain U is affiliated with rank v_i.

Here, we will calculate the membership to "P" for each attribute of U. The membership function of "h, p, b, r" are given as follows:

$$r_h = 1.0 / e^h \tag{1}$$

$$r_p = 1.0 / \log(p + 0.1) \tag{2}$$

$$r_b = 1.0 / \log(b + 0.1) \tag{3}$$

$$r_r = 1.0 / (1 + e^{-r/50.0}) \tag{4}$$

Then, the matrix R can be presented as the column vector for one path:

$$R = \begin{pmatrix} r_h \\ r_p \\ r_b \\ r_r \end{pmatrix}$$

4) The determination of weight vector $A = (a_1, a_2, \cdots, a_n)$. A represents the importance degree of the factor in domain U. For path load balancing ,A is defined as (0.4,0.15,0.15,0.3),where the length of the path is more important, the workload of critical switch and critical link share the left.

5) A is combined with R to gain the evaluation results with the composite operator.

$$B = (b_1, b_2, \cdots, b_m)$$

$$B = AOR = (a_1, a_2, \cdots, a_n) O \begin{pmatrix} r_{11} & \cdots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{n1} & \cdots & r_{nm} \end{pmatrix}$$

Here, for K paths, the final formula is:

$$B = (b_1, b_2, \dots, b_K)$$

$$= (0.4, 0.15, 0.15, 0.3) * \begin{pmatrix} r_{h1} & r_{h2} & \dots & r_{hk} \\ r_{p1} & r_{p2} & \dots & r_{pk} \\ r_{b1} & r_{b2} & \dots & r_{bk} \\ r_{r1} & r_{r2} & \dots & r_{rk} \end{pmatrix}$$

Where $b_i = 0.4 * r_{hi} + 0.15 * r_{pi} + 0.15 * r_{bi} + 0.3 * r_{ri}$, and b_i is the final evaluation value of path i.

IV. PATH LOAD BALANCING MECHANISM

Our SDN-based solution for path load balancing follows the layer architecture of SDN which consists of separated control plane and data plane, the framework of this solution is showed in Fig.1. The paths which transmit network traffic from source node to destination node should be the optimal for path load balancing. Meanwhile, the path correctness should be ensured for making full use of network resources. Therefore, the controller makes decisions by using the path evaluation model to select the paths which are conducive to path load-balancing. It is composed by Data Collection Module, Path Evaluation Module, Flow Table Installation Module.

A. *Data Collection Module*

1) Topology probe

The topology probe can help the controller have knowledge of host position with the global network view. It is realized by the links probe which is completed with transmitting LLDP data packets. The link events are triggered when the links are built or broken between the nodes.

2) Traffic data collection

Through periodically gathering statistical information from OpenFlow switches, the packet count and byte count forwarded by switch and the corresponding port forwarding rate are collected.

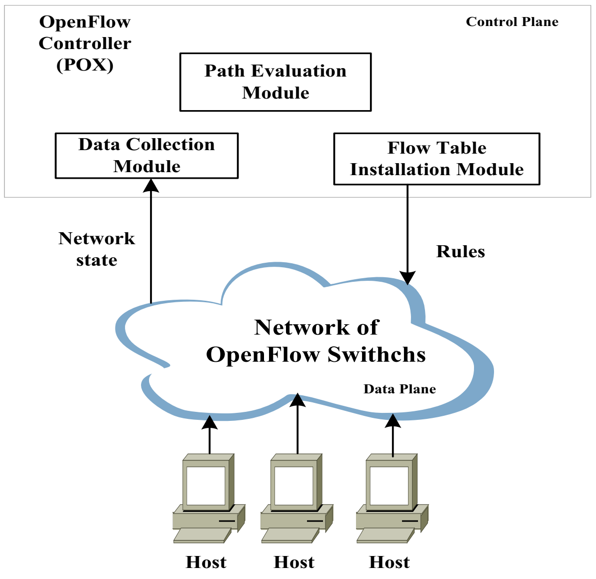

Fig. 1. SDN-based path load balancing framework

B. *Path Evaluation Module*

This module is constituted with Top-K paths algorithm and fuzzy synthetic evaluation algorithm.

1) Top-K paths algorithm

Since there may be many redundant paths in the network, the controller need firstly select the Top-K optimal paths to reduce the computational complexity in synthetically evaluating the paths in real time. It is divided as the initial mode and periodic mode.

In the initial mode, the Top-K shortest paths algorithm is implemented based on Floyd algorithm with the measure factor of hop count, which is significant to the network.

In the periodic mode, it mainly uses the synthetic evaluation method to choose Top-K paths at regular intervals. The method will be described in detail later.

The pseudo-code of the Top-K paths algorithm is shown in the Algorithm 1.

```
Algorithm1: Get the Top-K paths
initial mode :Get the Top-K shortest paths
Assuming that the Floyd-based shortest paths are acquired between each node, S and T represent two network nodes.
The array A[k] stores the K paths between S node and T node and A[0] apparently represents the first path.
1.  A[0] is the shortest path from S to T
2.  For each k in {1,2,…,K-1}
3.      For each i in {0,1,2,…,length(A[k-1])-2}
4.          nodeA equals to A[k-1].node(i)
5.              For each j in {0,1,2,…,k-1}
6.                  nodeB equals to A[j].node(i)
7.                      If nodeA equals to nodeB
8.                          The distance between nodeA and A[j].node(i+1) equals to infinity
9.                      EndIF
10.         EndFor
11.     S[i] is the shortest path from nodeA to T
12.     R[i] is in A[k-1] from S to nodeA
13.     B[i] equals to sum of R[i]  and S[i]
14.   EndFor
15. A[k] is the minimize length path amongst all B[i]
16. Restore the distance between nodeA and nodeB to original value
17. EndFor

periodic  mode :Get the Top-K optimal paths
    evaluate each path as similar as Algorithm2 for all the available paths,
    and select K paths by the evaluation value  every 5 minutes;
```

2) Fuzzy synthetic evaluation algorithm

The fuzzy synthetic evaluation algorithm is used for evaluating the Top-K optimal paths. The characteristic indicators are hop count, packet count and byte count forwarded by switch, port forwarding rate. They are collected from the global information of network, which includes the information of nodes and links. The weight vector is A(0.4,0.15,0.15,0.3) ,whose elements correspond to hop count, packet count, byte count and port forwarding rate respectively. Then, the membership functions of four factors are defined respectively. Finally, the weight vector A is combined with the matrix R by composite operator to achieve the evaluation results. The final evaluation score suggests the best path.

The pseudo-code of evaluating paths is shown in the Algorithm2.

```
Algorithm2: Synthetic  Evaluate Paths
Input: the Top-K paths K_PATH
Output: the optimal path BEST_PATH
1.  Define the factor domain U(hops, bytes, packets, port_rate) and
```

```
    rank domain V(P) for path
2.  For each i in {0, 1, 2, ..., K-1}
3.      Assign the value of length(K_PATH[i]) after critical
        calculation to hops[i]
4.      Assign the value of total bytes that flow into K_PATH[i] after
        critical calculation to bytes[i]
5.      Assign the value of total packets that flow into K_PATH[i]
        after critical calculation to packets[i]
6.      Assign the value of transfer rate in K_PATH[i] after critical
        calculation to port_rate[i]
7.  EndFor
8.  For each j in {0, 1, 2, ..., K-1}
9.      hops[j] transformed by 1.0 /exp(hops[j])
10.     bytes[j] transformed by 1.0 / log(bytes[j] + 0.1)
11.     packets[j] transformed by 1.0 / log(packets[j] + 0.1)
12.     port_rate[j] transformed by 1.0 / (1 + exp(-port_rate[i]/50.0))
13. EndFor
14. Define weight vector A(weight_hops, weight_bytes,
    weight_packets, weight_port_rate)
15. For each k in {0, 1, 2, ..., K-1}
16.     Assign temp_hops[k] equals to a value through composite
        operation between weight_hops and hops[k]
17.     Assign temp_bytes[k] equals to a value through composite
        operation between weight_bytes and bytes[k]
18.     Assign temp_packets[k] equals to a value through
        composite operation between weight_packets and packets[k]
19.     Assign temp_port_rate[k] equals to a value through
        composite operation between weight_port_rate and port_rate[k]
20.     Assign score[k] equals to the sum of temp_hops[k]
        temp_bytes[k] temp_packets[k] and temp_port_rate[k]
21. EndFor
22. BEST_PTAH equals to the max value among all score
```

C. Flow Table Installation Module

The forwarding rules are generated by the central controller according to the path evaluation results. They are installed to the openflow-enabled switches to direct transmitting the network traffic. Considering that the network traffic and topology are unpredictable, the openflow table will be dynamically updated to keep consistent with the strategy of path load balancing. On one hand, the paths are timely evaluated by the mechanism described previously when the new flow arrives or the network changes, and the optimal path is installed to the corresponding switches. On the other hand, the global statistical information is timely updated and the flow table is configured to delete timeout entries.

V. SIMULATION AND ANALYSIS

In order to evaluate the proposed solution of SDN-based path load-balancing, we have implemented a component in POX platform and used Mininet [19,20] to simulate the network topology. POX is a platform for the rapid development and prototyping of network control software using Python. Mininet can help users quickly create a realistic virtual network, running a collection of hosts, switches and network links on a single machine. Mininet provides a lightweight test bed for developing OpenFlow applications. The controller invokes the components to perform corresponding actions when the data packets fail to match with forwarding rules. Mininet is running in the form of virtual machine in an experimental host, which depends on the Linux kernel. Therefore, the performance of Mininet is influenced by the host configuration. These detailed parameters of the host configuration is listed in Table Ⅰ.

TABLE I. PARAMETERS OF THE HOST CONFIGURATION

Parameters	value
Physical machine	Intel Core i3-2100 3.1GHz processor 4GB memory
Operating system	Linux-3.5.0-17-generic-x86_64-with-Ubunu-12.10
Virtual machine	Virtual Box

We have simulated effectiveness test and robustness test to verify the solution performance. In order to have good stability and comparability of the experiment, the simulation environment is designed with the topology of 8 OpenFlow switches and 10 user terminal, as shown in Fig.2. Switch-k represents an OpenFlow switch whose switch ID is k; Host-k represents a host which is connected to the switch and its host ID is k. Meanwhile, the proposed solution for path load balancing is also compared with solution of the shortest paths to analyze performance.

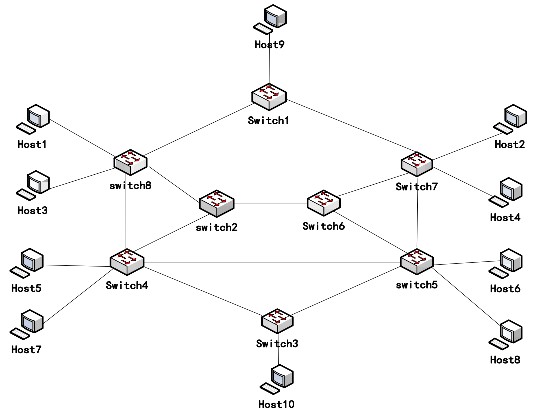

Fig. 2. Topology of Simulation

A. Effectiveness

This experiment has been implemented to prove the effectiveness of the links evaluation module. The Host5 and Host2 are chosen as the observing subject. Firstly, we use the Top-K shortest paths algorithm to select three shortest paths, then, the three shortest paths are dynamically evaluated by fuzzy comprehensive evaluation model with the changing status of network. At the beginning, the evaluation value is mainly determined by the hop count. Then, the Host6 sends ping packets to Host4 to inject a small count of workload between Swithch5 and Switch7. The Host5 sends UDP packets to Host8 to occupy the bandwidth resources between Switch4 and Switch5. The evaluation values of three paths begin to vary with the changing link status. For path one, its two links and three switches are all affected, so its evaluation value declined most. For path three, only switch 7 is implicated, so the evaluation value just dropped a little. The experimental results are shown in Fig.3. Finally, the new optimal path will be chosen to transmit traffic load to acquire path load-balancing for the whole network.

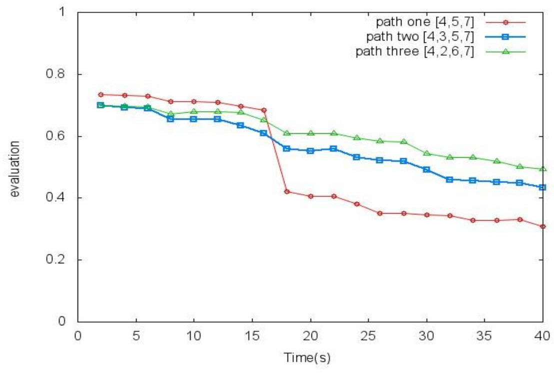

Fig. 3. The evaluation values of paths

The experimental results show that the proposed solution of path load-balancing is effective and can dynamically change path with the traffic information of network nodes and links.

B. Reliability

In order to prove the robustness of the proposed solution when the network nodes and links suffer from breakdown, we design the robustness experiment. The Host5 and Host6 are chosen as the observing subject when we manually break the link between Switch4 and Switch5. The experimental results are shown in Table II, it reflects that the traffic can restore after a transient pause due to the link failure. Since it takes a little time to detect the break, a few ping packets will be lost in the procedure. After that, the delay time of ping packet becomes normal quickly.

TABLE II. THE TRANSMISSION DELAY OF PACKETS

Host5 ping Host6: ping 10.0.0.6
64 bytes from 10.0.0.6: icmp_req=1 ttl=64 time=0.050ms
64 bytes from 10.0.0.6: icmp_req=2 ttl=64 time=0.056ms
64 bytes from 10.0.0.6: icmp_req=3 ttl=64 time=0.049ms
64 bytes from 10.0.0.6: icmp_req=4 ttl=64 time=0.052ms
64 bytes from 10.0.0.6: icmp_req=5 ttl=64 time=0.058ms
64 bytes from 10.0.0.6: icmp_req=6 ttl=64 time=0.065ms
64 bytes from 10.0.0.6: icmp_req=7 ttl=64 time=0.053ms
Break the link between Switch4 and Switch5
64 bytes from 10.0.0.6: icmp_req=15 ttl=64 time=164ms
64 bytes from 10.0.0.6: icmp_req=16 ttl=64 time=0.511ms
64 bytes from 10.0.0.6: icmp_req=17 ttl=64 time=0.059ms
64 bytes from 10.0.0.6: icmp_req=18 ttl=64 time=0.062ms
64 bytes from 10.0.0.6: icmp_req=19 ttl=64 time=0.051ms
64 bytes from 10.0.0.6: icmp_req=20 ttl=64 time=0.050ms

The experimental results show that the controller can detect the link fault and timely select another path (e.g., switch4,switch3,switch5) to transmit data packets to cope with the change of network status. The proposed solution of path load-balancing enhances the network reliability, it can instantaneously switch the paths to avoid breakdown caused by the failure of single path.

C. Analysis of Efficiency

In order to analysis the efficiency of proposed solution, the shortest path solution is taken as comparison. The Host1 and Host3 are chosen as the observing subject in the network topology with 9 OpenFlow switches and 8 user terminals, as shown in Fig. 4. The RTT is tested as the experimental result, as shown in Fig. 5. It shows that the RTT fluctuates violently with the shortest path solution when the path is becoming more and more crowded. However, the RTT remains stable with FESM solution, since the traffic will be transferred to the new best path timely.

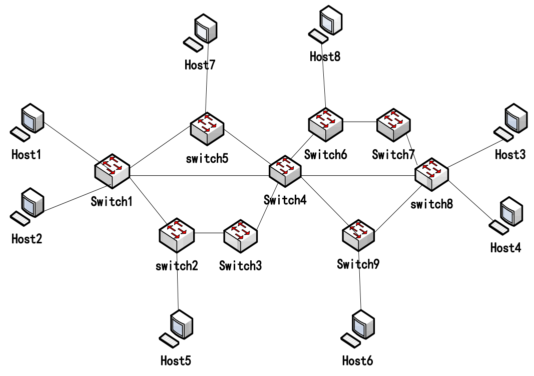

Fig. 4. Topology of comparing performance

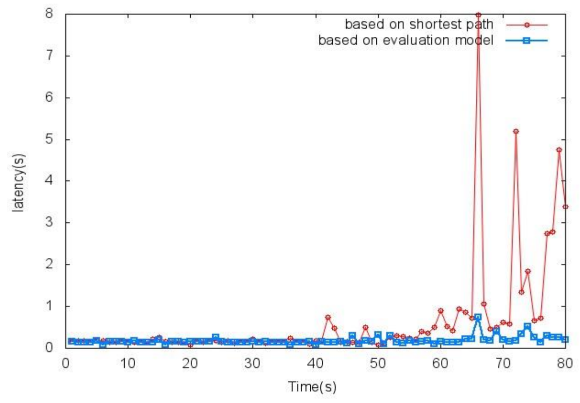

Fig. 5. RTT of two solutions

The experimental results reveal that the proposed solution of path load-balancing is better to adjust the transmitting paths and guarantee the quality of packet transmission.

VI. DISCUSSIONS

This section discusses several issues that were not previously mentioned.

The value of K: In fact, the value of K of the Top-K paths can be decided by the network manager. In this implementation, the number of K is given as 3.

The weight of factors: In this paper, we assign the weight with constant value. In the future, they can be dynamic with the change of the path status.

The choice of critical node and link: We assume all the nodes and links have the same capability, this will simplify the procedure of getting the critical node and link. Although it may not be consistent with the real network, it doesn't impact the main idea of this paper.

VII. CONCLUSION

This paper has presented a solution of path load-balancing based on SDN to increase the utilization and reliability of network paths. The fuzzy synthetic evaluation model is adopted in this solution to dynamically select the optimal path in response to the change of the network status to transmit the network traffic. We have implemented this solution in the POX platform and conducted comprehensive experiments in the Mininet to evaluate the effectiveness, reliability and efficiency of our solution. The simulation results verify that the proposed solution can effectively adjust the transmitting paths and avoid the unexpected breakdown caused by failures of the path. The total network performance has also been verified to be improved as well. In the future, more sophisticated path routing algorithms, for example, the method of using multipath forwarding to leverage available network capacity according to application priority, will be combined with the mechanism described by this paper.

ACKNOWLEDGMENT

We thank Chunjia Liu,Qingqing He, Lingrui Li, Peng Liu and Pengfei Jin for fruitful discussions on the topic and active feedback on the experiments of this work.

This work is supported by the Director Fund(Grant No. CNIC_ZR_201204), and the Around Five Top Priorities of "One-Three-Five" Strategic Planning, CNIC(Grant No. CNIC_PY_1401), Chinese Academy of Sciences .

REFERENCES

[1] "Software-Defined Networking: the new norm for networks," White Paper, Open Networking Foundation, 2012.

[2] Haleplidis E, Denazis S, Koufopavlou O, Halpern J, Salim JH, "Software-defined networking: experimenting with the control to forwarding plane interface," In: Proceedings of european workshop software defined networking (EWSDN), pp 91–96, Oct 2012.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, et al, "Openflow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, April 2008.

[4] Wang R, Butnariu D, Rexford J, "OpenFlow-based server load balancing gone wild," In: Proceedings of the 11th USENIX conference on hot topics in management of internet, cloud, and enterprise networks and services, pp 12–12, Mar 2011.

[5] POX, https://openflow.stanford.edu/display/ONL/POX+Wiki

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. "NOX: Towards an operating system for networks," ACM SIGCOMM Computer Communications Review, 38(3), 2008.

[7] Floodlight, http://www.projectfloodlight.org/floodlight/

[8] OpenDaylight, http://www.opendaylight.org/

[9] Handigol N, Seetharaman S, Flajslik M, McKeown N, Johari R, "Plug-n-serve: loadbalancing web traffic using openflow," In: Proceedings of demo at ACM SIGCOMM, Aug 2009.

[10] Koerner M, Kao O, "Multiple service load-balancing with OpenFlow," In: Proceedings of the 13th international conference on high performance switching and routing, pp 210–214, 24–27 Jun 2012.

[11] Uppal H, Brandon D, "OpenFlow based load balancing," In: Proceedings of CSE561: networking. project report. University of Washington, Spring 2010.

[12] Li-Der Chou, Yao-Tsung Yang, Yuan-Mao Hong, Jhih-Kai Hu, Bill Jean, "A Genetic-based load balancing algorithm in OpenFlow Network," Advanced Technologies, Embedded and Multimedia for Human-centric Computing, Lecture Notes in Electrical Engineering , vol. 260, pp 411-417, 2014.

[13] S. Jain, et al, "B4:experience with a globally-deployed software defined WAN", SIGCOMM, 2013.

[14] Chi-Yao Hong, Srikanth Kandula,Ratul Mahajan,Ming Zhang,Vijay Gill ,Mohan Nanduri,Roger Wattenhofer, "Achieving High Utilization with Software-Driven WAN", SIGCOMM, 2013.

[15] Hui Long, Yao Shen, Minyi Guo, Feilong Tang, "LABERIO:Dynamic load-balanced routing in OpenFlow-enabled networks," In: Proceeding of the 27th international conference on advanced information networking and applications, pp 290-197, 25-28 March 2013.

[16] Bellman R E, Zadeh L A. Decision-making in a fuzzy environment [J]. Management Science, 1970, 17(4): B-141.

[17] D. R. Shier. "On algorithms for finding the k shortest paths in a net" [C]. Networks,Volume 9, Issue 3, pages 195-214, Autumn, 1979.

[18] R. Floyd. Algorithm 97, shortest path. Comm. ACM, 1962, 5:345.

[19] Mininet, http://mininet.org/

[20] Lantz B, Heller B, McKeown N. A network in a laptop: rapid prototyping for SoftwareDefined Networks [C]. proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010: 19.