問題:在早期大部分 static routing path 使用 OpenFlow
, 往往這樣的方式在網路配置過程中是最不理想的。

解決:在一個網路環境中, 如何 load balanced 使用 OpenFlow protocol?假設發現一個靜態(RR(Round Robin) and LOBUS
)路由路徑在初始化步驟, 而靜態路由路徑經常遭受以來表現不佳的網路配置過程中可能會改變數據傳輸。**為了解決這個問題提出了LABERIO(LoAd-BalancEd Routing wIth OpenFlow)減少13%傳輸時間。**

in OpenFlow-enabled networks

o, Feilong Tang
, Shanghai Jiao Tong University
Shanghai, China
*Corresponding Author
{longhui617, feilongtang8}@gmail.com {yshen, guo-my}@cs.sjtu.edu.cn

LOBUS(LOad-Balancing over UnStructured networks)

***Abstract*—In data center networks, how to balance workloads is a key issue with the fast growth of network applications. OpenFlow protocol, which is a competitive candidate for solving the problem, provides each user the programmatic control for specific flows, so as to determine their paths through a network. However, existing solutions based on OpenFlow only try to find a static routing path during initialization step while the static routing path often suffers from poor performance since the network configuration may change during the data transmission. To solve the problem, this paper proposes LABERIO, a novel path-switching algorithm, to balance the traffic dynamically during the transmission. Experiments on two kinds of network architectures demonstrate that LABERIO outperform other typical load balancing algorithms like Round Robin and LOBUS by reducing up to 13% transmission time.**

Keywords-OpenFlow; load balancing; path selection; fat-tree topology;

I. INTRODUCTION

Load balancing is a very hot issue of high importance in traffic management field. The purpose of load balancing in a network is to distribute traffic evenly among multiple paths, thus make it able to process more data flows using less time. To avoid congestion on a server, many datacenters use load-balancer hardware devices to help distribute network traffic across multiple machines. However, these devices are often too expensive to be widely used. The emergence of OpenFlow technology brings about an effective and affordable solution to control the network traffic. It enables the software running on multiple routers to determine the path of network packets through the network. The first OpenFlow controller, named NOX [6], has been successfully used to separate the packets control from the forwarding, so as to allow for more sophisticated traffic management rather than just use access control lists (ACLs) and routing protocols [1]. Figure 1 shows the basic structure of a NOX-based OpenFlow network. In OpenFlow, switches are represented by flow tables with entries of the form <header : counters, actions>. For each packet matching a specified header, the counters are updated and the appropriate actions are taken. OpenFlow switches process traffic according to a limited number of rules on packet headers that are chosen and installed by an out-of-band controller [7]. In theory, per-flow rules could be used for load balancing. Therefore, it is possible to balance the traffic in an OpenFlow-enabled network.

Many research works have been done to balance workloads among [3, 5, 8, 18, 19] in OpenFlow networks. However, all those schemes focused on the initialization of the flows by leveraging path selection algorithms, while overlooked other issues that may get serious. For example, as the topology grows bigger and more complex, the time needed for the path selection at the initialization will be tremendously increased. On the other hand, these works will be of little help if something unexpected happens during the transmission, e.g. a link breakdown. Their proposals may avoid the aggravation for load imbalance, but cannot solve the overload issue when it happens.

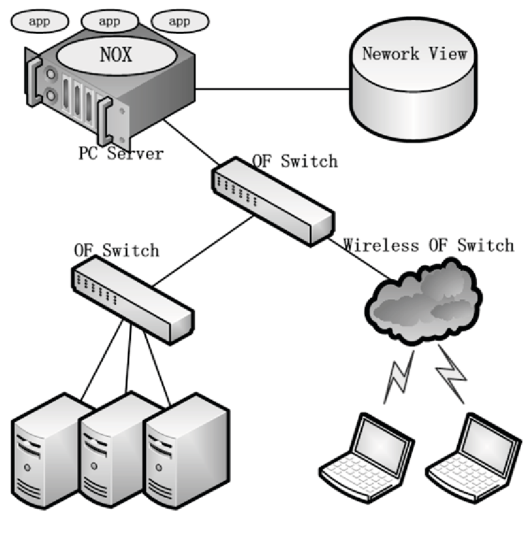


Figure 1. Components of a NOX-based network: OpenFlow (OF) switches, a server running a NOX controller process and a database.

With the above considerations, in this paper we apply a halfway switching strategy, which better utilizes the advantage of the OpenFlow controller as well as distributed network state adjustment. OpenFlow has a few limitations which have been discussed by Wang, et al. [5]. For example, it does not support hash-based routing [4] as a way to spread traffic over multiple paths so far. Due to this, we rely more on flexible load balancing algorithms to fit different imbalance situations, so as to avoid the inadequacy of the original OpenFlow protocol.

With the ultimate goal of dynamically achieving global load balance in an OpenFlow-enabled network, we leverage a novel integrated optimization algorithm, called LABERIO (LoAd-BalancEd Routing wIth OpenFlow). It is designed to improve the overall file transmission performance, to minimize latency and response time as well as to maximize the network throughput by better utilizing available resources.

To the best of our knowledge, this is the first work to do path-switching in the midway of a flow transmission within an OpenFlow network. The main contributions of this paper are:

- We developed LABERIO, a group of routing algorithms that theoretically achieve better performance on maximizing the throughput while reducing the total transmission time.
- We described an implementation for LABERIO in two different environments: a non-blocking full-populated network topology and a typical fat-tree network topology.
- Comparative study has been conducted and experiment results indicate that our algorithm generally performs better than other classical methods within multiple transmission modes. This will be declared later in Section V.

The rest of this paper is organized as follows: Section II provides a brief background on various existing routing algorithms, and compares them with LABERIO. Section III introduces the system models we use and defines the problem. Section IV describes the LABERIO routing algorithm and analyzes its behavior under different scenarios. Section V presents the experiments we have taken and evaluates the final results. Section VI summarizes and concludes this paper.

II. Related Work

Nowadays, load balancing issues and software defined networking (SDN) [9] have been extensively studied in distributed computing network.

Load balancing. A few papers analyze and contrast among different load balancing algorithms. In [10], Sharma et al. propose five algorithms: three of which are static load balancing methods and the other two are dynamic ones. Static load balancing refers to load balancing algorithms that distribute the traffic strictly based on a fixed set of rules according to the characteristics of the input traffic. It does not feedback real-time information about traffic amount on each link [2]. On the other hand, in a dynamic load balancing algorithm, the load distribution is decided according to the current processing rates and network conditions during transmission. But they didn't conduct experiments on those algorithms. Theoretical illustration alone is not that convincing. Recent proposals, such as TeXCP [12] and COPE [13], focused on dynamic approaches. However, they stress too much on the traffic splitting across multiple paths while lacking persuasive solutions to the packet reordering issue, which may happen frequently. LABERIO solves this problem by adding a sequence number to each subflow when every time the original path is being switched.

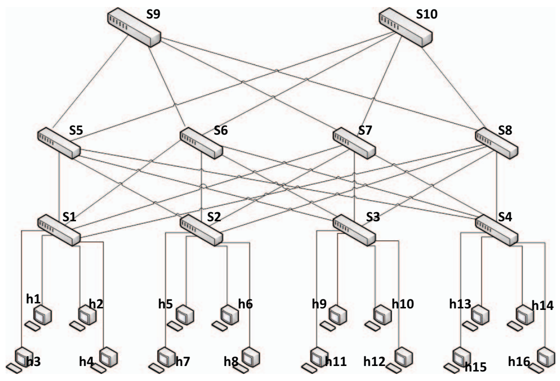


Figure 2. Model A: 3-level non-blocking fully populated network.

SDN architecture. SDN allows for quick experimenting and optimization of switching/routing policies. It can also be used for the external access to the innards of switches and routers that formerly were closed and proprietary. OpenFlow is a leading SDN architecture. In order to balance the load, Handigol et al. [3] provide a LOBUS algorithm, which simply applies greedy selection strategy to pick the (server, path) pair that yields the least total response time at every request. Anitha et al. propose an idea similar to the usage of OpenFlow switch flow table, which applies a load table on the Dispatcher node to record the change of state and then applies corresponding transfer policy [11]. All the above papers are trying to come up with a pervasive solution so as to timely balance the load at a global view. However, they neglected the unpredictable changes of the load status on tons of links. This will have even greater effect when the distributed network grows bigger and bigger.

III. Network Model and Problem Formulation

In this section, we firstly present the network model and then formulate the problem.

A. Network Model

The general network topology (model A) considered in our environment, consists of 2 core switches on 1st level, 4 Aggr (aggregation) switches on the 2nd level, and 4 ToR (top-of-rack) switches on the 3rd level, as shown in Figure 2. Sixteen end hosts are being connected to this network, titled from h_1 to h_{16}.

Another model (model B) which is frequently used in existing networks is the fat-tree topology. The fat-tree topology has many properties that make it attractive for large scale interconnects and system area networks. This topology structure is able to be expanded for different scales and it

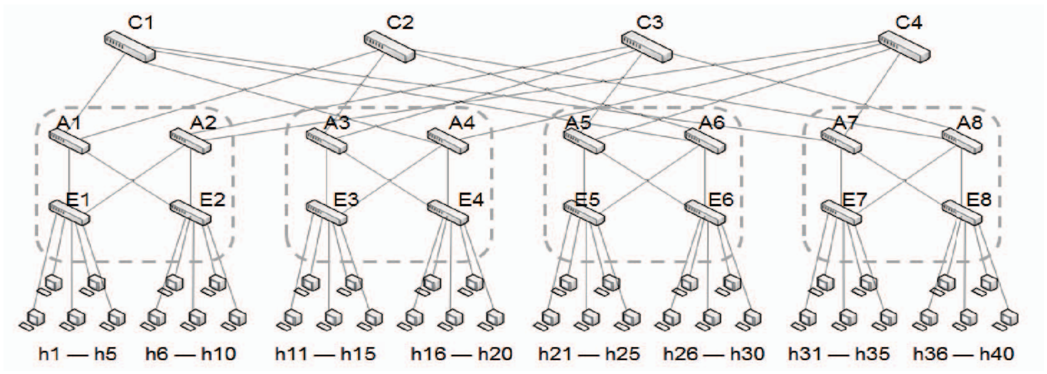


Figure 3. Model B: 3-level fat-tree network topology.

also provides redundant paths between two processing nodes [14, 15, 16].

Before delving into the detailed formulations of the load balancing problem on network traffic, we assume:

- QoS data flow is measured by the minimum bandwidth requirement.
- Every link L_i can transmit data at its capacity rate C_i, either one-way or double-way.
- All switches in the network are 4*4 switches which support OpenFlow protocols.
- Once a flow is being moved to another path, the divided subflows do not need to arrive in the original sequence order. Every subflow is marked by a sequence number to indicate its position in the original flow, which will help reorder the subflows after they all have reached the destination end host.
- End host will set a priority weight value for a flow before its transmission. This is done according to the urgency and importance of the flow as well as the end host's service priority.

To solve this problem, we divide our work into two phases: end host scheduling phase, and load-balanced routing phase, as described in part B.

B. Process description

In this sub-section, we will introduce the process of our network transmission, which mainly consists of two steps.

1) End Host Scheduling

For the initial flow scheduling mechanism, we come up with a Largest Weight First Served (LWFS) algorithm. It means, flows are scheduled based on the priority weight. The flow who gains the largest Priority Weight (PW) (due to the initial configuration on end hosts), will rank first to be served in the next time slot. To be more specific, we elaborate it with Model A in Figure 2. Suppose that end hosts *h1*, *h5* and *h10* simultaneously send transmission requests to the central controller at the beginning of time slot N. The central controller will first look into the Priority Weight Table (PWT) to find out the largest PW value, and pick out the corresponding flow request. Furthermore, if some coincidence happened, such as, the PW of two flows are of the same value, we would randomly choose one.

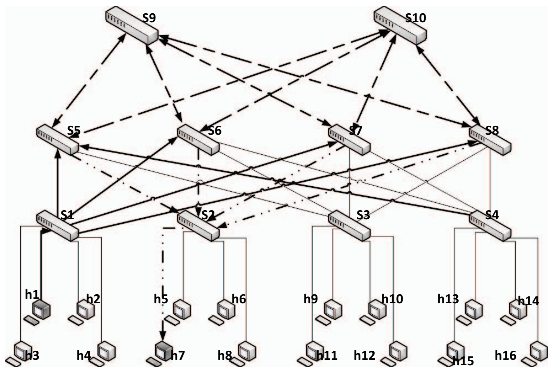


Figure 4. Links affected by the transmission task.

2) Load-balanced Routing

For the data preparation of load-balanced routing procedure, we need to create and maintain two important tables to store the basic information of our network. They are ToR Switch-to-ToR Switch Paths Table (S2SPT) and Load Allocation Table (LAT).

In the S2SPT, every end host connects to a unique ToR switch, so every row records all the possible paths from a certain source end host to every other end host. Similarly, every column records all the possible paths from every other end host to a certain destination end host. Thus, the S2SPT provides all paths information of every <*src, dst*> pair. This table may become bigger and bigger as the network grows. But since it can be generated offline and will be used as a look up table, the complexity is not a problem. We set the number of hops upper bounded by 4.

In the LAT, every row is indexed with a <*src, dst*> pair, and every column specifies a time slot interval. Every value in the LAT block indicates the corresponding remainder capacity of a specific link. Here we formulate it to be *C - Cost*. At the beginning, the LAT is initiated to be an all-C_i table, where C_i is the capacity of link L_i (in Kbps). At the very beginning of every time slot t_i, we update the values in the i^{st} column due to the scheduling strategy decided by the controller at time t_i. If a link is fully occupied, we set its LAT value to be zero. Similarly, when a link collapses, failure recovery mechanism would be achieved by setting the link capacity zero so that no more flows will be transmitting through this link. Then we continue to find the links which could be used if the transmission's requirement has still not been fully satisfied. To make it more explicit, we will discuss it under a certain scenario in section 3, where end host h_1 is going to transmit a flow that has a QoS requirement of no less than 150 Kbps to h_7 at the beginning of time slot 5.

TABLE I. ToR Switch-Pair's Available Path Table

S2SPT		**Destination ToR Switch**			
		S1	***S2***	***S3***	***S4***
Source ToR Switch	***S1***		{1,5,2}	{1,5,3}	{1,5,4}
			{1,6,2}	{1,6,3}	{1,6,4}
			{1,7,2}	{1,7,3}	{1,7,4}
			{1,8,2}	{1,8,3}	{1,8,4}
			{1,5,9,6,2}	{1,5,9,6,3}	{1,5,9,6,4}
			{1,5,9,7,2}	… …	
			{1,5,9,8,2}		… …
			{1,6,9,5,2}		
			{1,6,9,7,2}	… …	
			{1,6,9,8,2}		… …
			… …		
			{1,8,10,6,2}	… …	
			{1,8,10,7,2}	{1,8,10,7,3}	{1,8,10,7,4}
	S2	… …			
	S3		… …		
	S4			… …	

The corresponding S2SPT and LAT are respectively shown in TABLE I and TABLE II. Figure 4 shows how the current task affects the relative links on our network model A. Unidirectional solid lines indicate links serve for uphill only, while dotted ones for downhill. Bidirectional dotted lines are for links that are double-way available.

C. Problem Formulation

In this section, we will formally describe the Load Balance Routing Policy.

Definition 1: The one-hop load-balance detector parameter $\delta(t)$. For a given network topology, it is determined by the absolute gap between the overall average bandwidth occupancy and the real-time load on each link.

$$\delta(t) = \frac{\sum_1^N \left[\overline{\mathrm{load}_{i,j}(t)} - \mathrm{load}_{i,j}(t)\right]^2}{N} \qquad (1)$$

In (1), $\mathrm{load}_{i,j}(t)$ is the occupied bandwidth on link <*i, j*> at time *t*. *N* is the number of overall links in the network. Formula (1) is used to compute $\delta(t)$ in every clock cycle. We formulate $\delta(t)$ in the form of variance. In statistics, variance is standardly used to measure the degree of fluctuation of a random variable [21]. Here, we want to use this meaning to take a measure of the network's transmission balance status. So that the degree of network equilibrium at time *t* is negatively correlated to $\delta(t)$. This is one of the key parameters in our work because we are going to use it as a trigger of our balancing algorithms.

Definition 2: The one-hop LABERIO trigger threshold δ^*. For a given network model, it is decided by experiments under a fixed topology scenario. It varies according to the complexity and topology of specific network. We will elaborate this later in Section V. If $\delta(t) > \delta^*$, the controller will initiate the load-balance scheduling.

Basically, we are going to move the flow which occupies the largest amount of bandwidth on the most congested link, to some other available path.

Definition 3: The real-time bandwidth utilization rate $\eta(t)$ is measured by the ratio of the overall occupied bandwidth to the total capacity of all links.

$$\eta(t) = \frac{\sum_1^N \mathrm{load}_{i,j}(t)}{\mathrm{MAX_{LOAD}} * \mathrm{N}} * 100\% \qquad (2)$$

Here $\mathrm{MAX_{LOAD}}$ stands for the physical upper bound bandwidth on each link. The value of $\eta(t)$ can intuitively reflect the effectiveness of the applied load-balancing algorithm.

TABLE II. Process of LAT value updating

LAT		**Time Slot**		
		t0	***t5-***	***t5+***
Link	< s1, s5 >	750	**225**	**75**
	< s1, s6 >	750	**97.5**	97.5
	< s1, s7 >	750	**120**	120
	< s1, s8 >	750	165	165
	< s5, s2 >	750	**0**	0
	< s5, s9 >	750	600	**450**
	< s5, s10>	750	187.5	187.5
	< s6, s2 >	750	**133.5**	133.5
	< s6, s9 >	750	150	150
	< s6, s10>	750	375	375
	< s7, s2 >	750	420	**270**
	< s7, s9 >	750	750	**600**
	< s7, s10>	750	225	225
	< s8, s2 >	750	**75**	75
	< s8, s9 >	750	675	675
	< s8, s10>	750	525	525

a. During the packets transmission from h1 to h7, t0 is the initial state, t5- stands for the state before flow routing at t5, and t5+ for after flow routing at t5. Figures are of KB unit.

IV. Load-Balanced Routing Algorithm

Since the load needs to be balanced during transmissions of flows, we design the algorithm LABERIO as to find alternative links for the busy ones. With this algorithm, the workloads are balanced and a higher throughput is achieved. In this section, we first give out the algorithm applicable for the fully populated network model, then revised it to fit more flexible situations.

Initial path selection. When the controller received a new flow's request from end host h_1 to end host h_7, it will first compute an initial temporarily optimal path for the flow to take. All paths from h_1 to h_7 are available in S2SPT, in the cell at the intersection of *Col. s1*, *Row. s2* (where *s1* and *s2* are the ToR switches directly connected to h_1 and h_7, respectively). To decide which path to take, we apply the max-min remainder capacity strategy (MMRCS), which is based on the widely used max-min policy [20].

Load balancing. During the file transmission, we monitor the network status and apply LABERIO algorithm when necessary.

Single-Hop LABERIO Algorithm

```
Main process:
1.  while $\delta(t) > \delta^*$
2.      find the busiest link <i, j>, set our object flow to
        be the biggest flow $f_k$ on this link;
3.          find the substitute path of $f_k$ from head of <i,
            j> to end of <i, j>, i.e., $i \to m \to \dots \to j$;
4.              find the lightest path $P_l$ among $(P_1, P_2,$
                $P_3, \dots)$, where $p_1, p_2, p_3 \dots$ all provide a
                possible connection from i to j, and l is
                the light index;
5.              end
6.          end
7.      end
8.  end while
```

For example, if we detect the current network is in unbalanced status, i.e., $\delta(t) > \delta^*$, and Link <*s2, s5*> is the busiest one, we will find out the biggest flow on this link as the object flow. If there are three flows on Link <*s2, s5*> now, they are in following conditions: f_1 (*s2* → *s5*, 40%), f_2 (*s5* → *s2*, 20%), f_3 (*s2* → *s5*, 30%). Obviously, f_1 consumes the most bandwidth on Link <*s4, s6*>, so it is our object to move it onto another path in the following steps. In our experiment, the remainder bandwidth of a specific link can be obtained by referring to the Load Allocation Table (LAT). To find available substitute paths, controller just needs to look up values in the S2S Path Table (switch to switch).

For δ^*, before we start the scheduling algorithm, we compute the overall network throughput at different values of δ^* with our proposed algorithm, and plot the throughput change curve according to different values of δ^*, then observe the relation between the two variables and choose the δ^* of highest throughput.

However, in real experiment, we may find that some flow has been frequently switched so that the total number of hops is heavily increased. This will lead to an indispensable surge in the overall transmission time. This definitely disobeys our original intention for applying the load balancing algorithm. In that case, we revise the previous version of LABERIO, mark every flow by another flag, *h*, indicating the history of hop increment of this flow. Every flow's hop increment parameter h will be initialized to be zero at the beginning. Once a flow has triggered the LABERIO, that is to say, it has been switched to another substitute path (for example, from 1 hop to 3 hops), the hop increment parameter h will be updated: $h = h + 2$.

Then we update the previous algorithm to make it more robust, as shown below.

Revised Single-Hop LABERIO Algorithm

```
Main process:
1.  while $\delta(t) > \delta^*$
2.      find the busiest link <i, j>, set our object flow to
        be the biggest flow $f_k$ on this link;
3.          while $h_{f_k} > 3$ (3 is our estimated threshold
            for the hop increment limit);
4.              Drop it and set our object flow to be
                the next biggest flow $f_k$ on<i, j>;
5.          end while
6.          find the substitute path of $f_k$ from head of <i,
            j> to end of <i, j>, i.e., $i \to m \to \dots \to j$;
7.              find the lightest path $P_l$ among $(P_1, P_2,$
                $P_3, \dots)$, where $p_1, p_2, p_3 \dots$ all provide a
                possible connection from i to j, and l is
                the light index;
8.              end
            end
9.      end
10. end while
```

In another case, we consider a fat-tree topology. As the Figure 3 indicates, when flows are uniformly distributed among all links, we give up the one-hop-substitute method, since the substitute path here at least increase 4 hops compared to the original one. Here we decide to switch the object flow onto a totally new path, and name it *Multi-Hop LABERIO.*

In multi-hop LABERIO, we scan the usage of each link at a frequency τ (say 1000 ms). Of every interval τ, the overloaded hops (measured by the reminder bandwidth on them) will be picked out, if exist, and be put into a set Σ. Then we find out the flow which covers the most hops in Σ

and set it as our object flow for path switching. Switching strategy here is at a higher level, working on end-to-end path. Once a flow, (sent from *h1* to *h20*), is marked as the object flow, we move it onto another path from end switch to end switch, (*E1* to *E4*). The new path should satisfy the following conditions (3):

- During the latest τ load detection, the available or free bandwidth on the busiest hop of this path should be the maximum among all available paths from *E1* to *E4*;
- During the latest τ, no flow has been switched onto this path.

In order to better illustrate LABERIO in this case, we add the following definition as a supplement to those in part C of Section III.

Definition 4: In network model B, the multi-hop load-balance object set $\sum(t)$ collects the top 10% busiest hops (hops that have the lowest reminder bandwidth on them at time t). $\delta(t)$, defined in (1), is reused as the launch signal for multi-hop LABERIO algorithm (as shown below).

When $\delta(t)$ exceeds the threshold δ^*, the elements in $\sum(t)$ will be examined to nail down the object flow.

Multi-Hop LABERIO Algorithm

```
Main process:
1. while $\delta(t) > \delta^* \&\& \sum(t) \neq \emptyset$
2.     find the flow $f_k$ which covers the largest subset of $\sum(t)$, set it to be our object flow;
3.         find $p_s$, the substitute path for $f_k$ from source to destination switch, i.e., $E_i \rightarrow \ldots \rightarrow E_j$, which satisfies the two presumptions in (3);
4.             switch the $f_k$ onto the new path $p_s$;
5.         end
6.     end
7. end while
```

Note that, our algorithms primarily concentrate on data flows in the network. We didn't consider the overhead generated by transferring control signals. We also assumed that all the control signals are synchronized and transmitted without perceivable latency. These two assumptions are reasonable because in OpenFlow network, the size of control flow is very small compared to that of the data flow.

V. Experment and Performance Evaluation

A. Evaluation Methodology

The performance of mid-way load-balanced routing can be measured by the actually delivered throughput under realistic conditions with the network model we have applied. We use experiment to evaluate the single-hop and multi-hop LABERIO algorithms. We apply the dynamic routing methods in two different three-level network models with the following procedure: initialization, routing, bandwidth monitoring and mid-way switching. We assume that the logic delay of calculation on link status and the time of response between the controller and local switches can be accommodated within one clock cycle all together.

We present results for two network size:

- Model A (Fully populated): four core switches * four pods. Each pod contains four switches, and each end switch directly links to five end hosts.
- Model B (Fat-tree): two core switches * four Aggr switches * four ToR switches. Each ToR switch directly links to four end hosts.

The base topologies are respectively modeled as the architecture illustrated in Figure 2 and Figure 3. For Model A, we use single-hop LABERIO algorithm since the number of one-hop substitute paths under a fully populated network is considerable. For the fat-tree topology (Model B) deployment, we use multiple virtual machines running on one server to represent the end hosts in one pod. We choose the revised multi-hop LABERIO routing algorithm as our experiments indicate it achieves better performance compared to the other one. The flow size and bandwidth upper-bound on each link are set by our experience.

We consider three different traffic patterns, uniform, semi-uniform and center-based. The first two communication modes differ in the way which *dst*-node is chosen for a given src-node. *Uniform* mode means the flows are initiated symmetrically among all hosts. *Semi-Uniform* means flows on the inter-pod links and flows on the intra-pod links are respectively evenly distributed. While the last pattern is similar to *hot-spot* traffic pattern where over 80% overall flow is sent out from one single node.

We have performed extensive experiments to study the performance of the proposed algorithms under the two network models explained in Section III. We compare our revised algorithm with two baseline algorithms, as defined as below:

1) *LOBUS.* This algorithm is a simple greedy selection algorithm, which:

a) Maintains running averages of the service time at each link.

b) Greedily pick the (host, path) pair that yields the lowest total response time for each request.

2) *Round Robin (RR).* This algorithm is one of the most classical static load balancing algorithms. In the round robin [17]:

a) Tasks are assigned evenly between all switches. Each new task is assigned to new available switch in round robin order.

b) The tasks allocation order is maintained on each switch's flow table locally independent of allocations from remote ones.

With equal workload round robin algorithm is expected to work well.

B. Experiment Setup

To emulate the desired traffic, we ran one central server to act as the controller. And we use *python* to modify the routing module in NOX to realize our algorithm. The number of additional servers for network model A and B is 10, 8, respectively. We set the bandwidth upper bound to be 750Kbps. Each single flow is assumed of the same size, 500M. OpenFlow switch is simulated by running *Open vSwitches* on servers. In model A, we built four virtual machines within each end vSwitch server; In B, there are five in each pod server.

C. Results

One major conclusion from our experiment is that the performance of revised LABERIO is closer to LOBUS and RR when we transmit flows symmetrically in pods internally. In the cases where the major traffic is distributed on the intra-pod hops, our algorithm outperforms RR and LOBUS at a notable degree. We also observe that LABERIO does achieve higher bandwidth utilization rate in the case of hot-spot traffic, but this comes at the cost of increased the number of total transmission hops and the system complexity. The experiment results are presented in Figure 5, 6, 7.

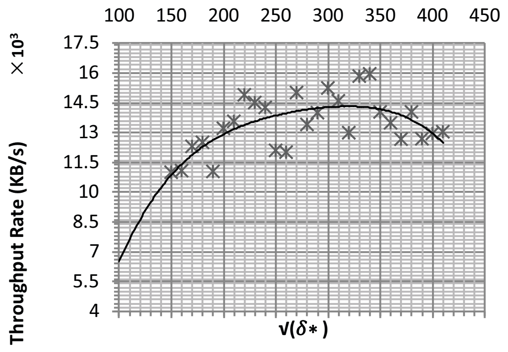


Figure 6. Relationship between one-hop LABERIO trigger and overall throughput.

Figure 6 presents the curve that helps us identify a proper value of δ^* for experiment on model A. For the convenience of data plotting, we use the square root $\sqrt{\delta^*}$ instead of δ^* itself. We also plot the trend line of the scattered curve, in order to better showing the optimal point of δ^*. Here we can clearly see that when $\sqrt{\delta^*}$ approximates to 340, the throughput rate of the overall network reaches a local maximum, 15945.8 Kbps, which is also a global maximum within the predefined bandwidth range.

The total time consumed by the pre-defined transmission tasks using three typical transmission modes is presented in the histogram in Figure 5, each with different algorithms we have illustrated in part A. We clearly observe that in center-based scenario, where the load is heavily asymmetrically distributed, LABERIO obviously brings down the completion time by 13% and 9% compared with RR and LOBUS, respectively.

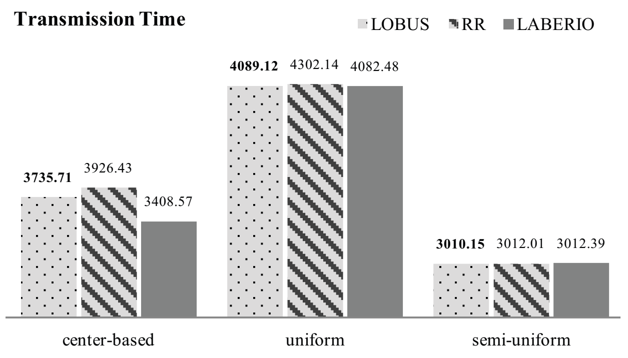


Figure 5. Performance comparison on total time.

Besides the overall transmission time, we also compare the bandwidth utilization rate, as defined in (2). Figure 7 (a) – (b) reflects the network status in Model A and (c), (d) are for Model B. Both Figure 7(a) and (c) studies the result based on an intra-pod symmetric transmission mode, which means, the flow is initiated between any pair of end hosts with ideally equal probability. In this case, we anticipate a relatively smooth curve in Figure 7(a) and (c) since the load is being well-distributed at the beginning. In another mode, which is opposite to the uniform one, one of the end hosts is set as the central server, who delivers files to all the other hosts. In Figure 7(b) and (d), LABERIO shows an outstanding advantage in this scenario. Hot spot in such kinds of three-level network is very likely to induce a global network imbalance. The rate fluctuation of LABERIO curve in (b) and (d) is much more wildly than that of LOBUS and RR. What's more, we also see an averagely higher bandwidth utilization rate of LABERIO than that of others, especially during the period of [0s, 2500s]. Thus, we can conclude that LABERIO has great potential in improving the load balancing in unstable networks. For inter-pod transmission mode in Model B, we did not design specific experiment. Since the load within a pod mostly concentrates on the links between end hosts and ToR switches, which can hardly be moved onto other paths without expensive time cost.

VI. Conclusion

In this work, we have designed a novel algorithm targeting at load balancing issue in an OpenFlow network. Although there are many existing algorithms for load balancing and routing strategy in SDN, they are not omnipotent when being placed to the large-scale distributed network because they fail to take into account the load crash in the middle of flows' transmission. We have proposed an efficient path switching algorithm to remediate the load imbalance issue generated during the transmission. Results of extensive experiments have shown that our algorithm do perform better than other alternative ones averagely.

ACKNOWLEDGMENT

This work was supported by the 863 Program of China (No.2011AA01A202), the National Natural Science Foundation of China (NSFC) with Grant Nos. 61272442, 61073148 and 61028005, and Key Basic Research Project of the Science and Technology Commission of Shanghai Municipality with Grant No. 12JC1405400.

REFERENCES

[1] Kate Greene "TR10: Software-Defined Networking," MIT Technology Review. Retrieved Oct. 7, 2011.

[2] "Static Load Balancing Implemented with Filters," White Paper

[3] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-Serve: Load-balancing web traffic using OpenFlow," Demo at ACM SIGCOMM, Aug. 2009.

[4] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, "Ensemble Routing for Datacenter Networks," In ACM ANCS, La Jolla, CA, 2010.

[5] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild," in Workshop on Hot-ICE, Mar. 2011.

[6] Natasha Gude, Teemu Koponen, Justin Pettit et al., "NOX: Towards an Operating System for Networks," SIGCOMM Comput. Commun. Rev., Vol. 38 (July 2008), pp. 105-110.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," SIGCOMM Comput. Commun. Rev., 2008.

[8] Carlos A. B. Macapuna, Christian Esteve Rothenberg, Maur´ıcio F. Magalhaes, "In-packet Bloom Filter Based Data Center Networking with distributed OpenFlow controllers," GLOBECOM Workshops, 2010.

[9] N McKeown, "Software-Defined Networking," INFOCOM keynote talk, Apr. 2009.

[10] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma. "Performance Analysis of Load Balancing Algorithms for cluster of Video on Demand Servers," IACC, 2011.

[11] T. N. Anitha, Dr. R. Balakrishna, "An Efficient and Scalable Content Based Dynamic Load Balancing Using Multiparameters on Load Aware Distributed Multi-Cluster Servers," IJEST, 2008.

[12] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive yet Stable Traffic Engineering," In SIGCOMM, 2005.

[13] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "Cope: Traffic Engineering in Dynamic Networks," In ACM SIGCOMM, 2006.

[14] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," Workshop on Communication Architecture for Clusters, pp. 1-8, Mar. 2007.

[15] Xin Yuan, Wickus Nienaber, Zhenhai Duan, Rami G. Melhem, "Oblivious Routing for Fat-tree Based System Area Networks with Uncertain Traffic Demands," In SIGMETRICS, 2007.

[16] Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance,Server-centric Network Architecture for Modular Data Center," In Proc. SIGCOMM, 2009.

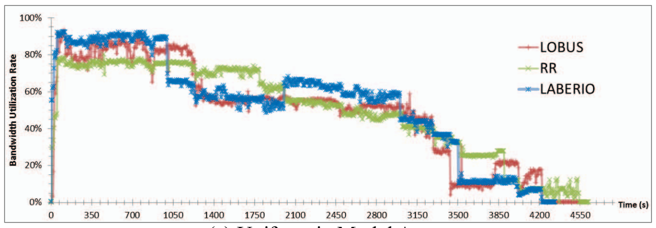
[17] Zhong Xu, Rong Huang, "Performance Study of Load Balancing Algorithms in Distributed Web Server Systems", CS213 Parallel and Distributed Processing Project Report.

[18] N. Handigol and etc., "Aster*x: Load-balancing web traffic over wide-area networks. GENI Engineering Conf. 9, 2010.

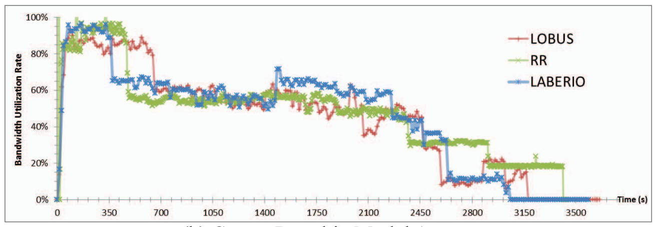
[19] Marc Koerner, Odej Kao, "Multiple Service Load-Balancing with OpenFlow," In Proc. HPSR, 2012, pp. 210-214.

[20] D. Bertsekas and R. Gallager, "Data Networks," Chapter 6, Prentice Hall, 1992.

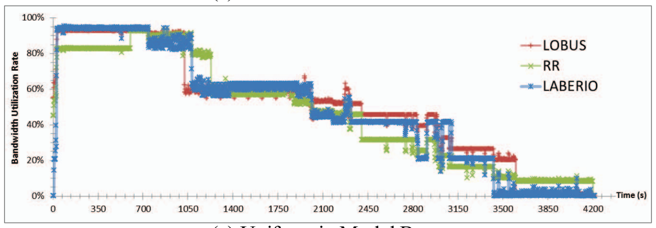
[21] S. Sancho, F. Ramirez and A. Suarez, "Analysis and reduction of the oscillator phase noise from the variance of the phase deviations, determined with harmonic balance," IEEE MTT-S, Atlanta (GA), USA, 2008.



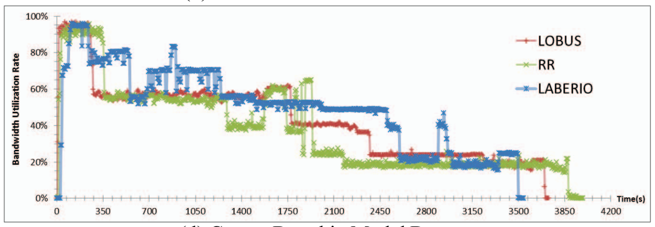
(a) Uniform in Model A



(b) Center-Based in Model A



(c) Uniform in Model B



(d) Center-Based in Model B

Figure 7. Performance comparison on bandwidth utilization.