Volume 8- Number 1- Winter 2016 (15-24)

SDTE: Software Defined Traffic Engineering for Improving Data Center Network Utilization

Marzieh Khoshbakht
The Faculty of Electrical and Computer Engineering
Tarbiat Modares University
Tehran, Iran
m.khoshbakht@modares.ac.ir

Mohammad Mahdi Tajiki
The Faculty of Electrical and Computer Engineering
Tarbiat Modares University
Tehran, Iran
mahdi.tajiki@TMU.ac.ir

Behzad Akbari
The Faculty of Electrical and Computer Engineering
Tarbiat Modares University
Tehran, Iran
b.akbari@TMU.ac.ir

Received: November 28, 2015- Accepted: March 12, 2016

Abstract--In recent years, several topologies with multiple-path between each pairs of end hosts for data center (DC) networks have been proposed. However, the path diversity is shown to be not enough to improve the network performance. Researches on the DC network measurements have shown that congestion occurs even when the average utilization of links is low, which means that some of the links are over-utilized while others are underutilized and have a considerable available bandwidth. Therefore, traffic engineering (TE) is necessary for proper distribution of the network load as well as exploiting the path diversity that is provided by new topologies. Current Equal Cost Multi Path (ECMP) based approaches are not efficient in lots of cases because numerous big flows may collide on the same path. The centralized solutions depend on the ability to predict the traffic pattern, which is not effective for unpredictable traffic patterns of data centers. In this paper, SDTE, an online software defined TE approach is proposed for cloud data centers. The proposed system does not depend on the ability to predict traffic pattern or the size of flows. SDTE exploits the PEFT routing algorithm to assign weights to links. SDTE is implemented within the OpenFlow framework. The evaluation shows that SDTE performs close to the optimal routing (average deviation is about 7%).

Key words: Software Defined Networking, Data Center Routing, Multipath Routing, Traffic Engineering, Cloud Computing, OpenFlow.

I. INTRODUCTION

Cloud computing [17] is one of the fast growing segments of IT industry in which services are highly available from anywhere/anytime. A data center (DC) is the underlying infrastructure that is used by Cloud. DCs are now an important part of the Internet that host a wide variety of applications and cloud-based services. With the expansion of Cloud, demands for cloud computing services grows. Consequently, it becomes critical for DC planners to address both current and future needs of cloud data centers [10]. The DC network architecture typically consists of routing and switching elements, that looks like a tree in which hosts are in the lowest layer, however, expensive non-commodity switches are in the higher layer(s). The difference in the cost of the commodity and non-commodity switches inclines the planners toward using lots of small commodity switches for building large-scale communication networks instead of using a few numbers of expensive ones. Therefore, several architectures with the aim of horizontally (rather than vertically) expansion of DCs have been proposed. These methods exploit a large number of inexpensive commodity switches to increase the aggregate bandwidth among the communicating hosts. These architectures are called multi-rooted tree as they provide a large number of parallel paths between servers. Lots of these kinds of topologies (e.g., Fat-tree shown in Figure 1) are based on clos topology [11].

It should be mentioned that even with a high available bandwidth, the utilization of the network is affected by the flow scheduling. Therefore, in order to exploit the capacity of path diversity provided by multi-rooted topologies, traffic must be engineered on all possible paths. Studies on the traffic patterns and measurements of DCs show that the current DC networks are under-utilized [18], [19] and [20]. Therefore, operators should optimize their network infrastructure before expanding their topologies or upgrading to new fabrics [10]. In the light of this fact, the reduction of congestion and load balancing on all reachable paths can improve the overall network performance along with maximization of the aggregate network utilization.

While many TE approaches have been suggested for the Internet, TE for DCs is still at the initial state. Two state-of-the-art solutions are the Equal-Cost-Multi-Path (ECMP) [8] and Valiant Load Balancing (VLB) [12] approaches. ECMP and VLB are not aware of link load. ECMP is a network congestion condition agnostic approach that splits the load across the available paths evenly. When a packet arrives, it is forwarded on the path that corresponds to a hash of selected fields of the packet's header: therefore, all the packets of a flow take the same path. Two or more big flows (flows that carry out large amount of data) can collide on their hash and be routed through one path which can lead to persistent congestion on some links while other links remain under-utilized [8].

However, several solutions including both centralized (e.g., Hedera [5], MicroTE [7], and Mahout [6]) and distributed solutions (e.g., MPTCP [24]) are provided that are load-aware. In the centralized solutions, the routing decisions are made by a global controller, while in the distributed solutions, the routing decisions are made by end-hosts or switches. MicroTE [7] is a system that adapts to the traffic variations through leveraging the partial predictability of the traffic matrix (TM). MicroTE relies on the traffic predictability and when a large portion of the traffic is predictable, it has a high efficiency; otherwise, it seamlessly shifts to using ECMP. First, it routes the predictable traffic optimally and then uses the weighted ECMP to route the unpredictable traffic [7]. Based on the studies [18, 19, and 20], DCs have a burst traffic pattern which is not predictable and leads weak performance of the TM prediction-based TE approaches like MicroTE.

Hedera [5] detects large flows at the edge switches and estimates the large flows demand. Hedera uses the placement algorithms to compute the proper paths for them. At the end, these paths are installed on the switches [5]. Hedera uses ECMP to schedule the small flows (short lived flows) and a centralized controller to schedule the large flows that exceed 10% of the host-NIC bandwidth. The problem is that in Hedera only flows exceeding 10% of the NIC bandwidth are considered as large flows. This cannot be an appropriate definition for the large flows because there are flows with a long lifetime transmitting at rates below 10% of the NIC bandwidth while such flows are never scheduled by Hedera. In Mahout [6], a shim layer on each end-host monitors the host flows in order to detect elephant flows (big flows). When the shim layer detects an elephant flow, it marks the subsequent packets of that flow using an in-band signaling mechanism. Then, switches tell Mahout Controller that there is an elephant flow through sending these packets to the Controller. Mahout Controller places the elephant flow on the least congested path. Mahout modifies the end-hosts and it is too complex to be implemented.

The mentioned approaches schedule the flows using ECMP by default, however, when traffic is predictable or elephant flows are detected, a series of simple and non-optimal algorithms are used. These approaches require modifications on switches and end-hosts. Multipath TCP [25, 31] is an end-host solution that splits a flow into sub-flows and balances the load across the sub-flows. Each sub-flow is similar to a regular TCP connection. The problem of end-host solutions is that they have to react to the congestion on the paths and rebalance the load due to the lack of a global view of the network.

Software Defined Networking (e.g. via OpenFlow [16]) as a new concept creates a new networking paradigm in which programming the network data path becomes possible. OpenFlow switch is based on an Ethernet switch with an internal flow table and a channel to an external controller which makes the routing decisions. Each flow table in the switch contains a set of flow entries and when a packet arrives, these entries are matched with the packet header's information. If a matching is made, the instructions associated with the specific flow entry will be executed, otherwise the packet will be forwarded to the controller through the OpenFlow channel. The controller can add, update, and delete the flow entries on the switches flow table using OpenFlow protocol [13].

Many proposals use software defined networking idea to improve traffic engineering. The Fibbing [1] controller injects fake nodes to create a fake topology in order to deceive the routers. Weighted Cost Multi-path (WCMP) [2] is a solution for weighted traffic hashing; it distributes traffic among all available paths in proportion to the available link capacity. WCMP assigns weights to each egress port in a multipath topology. These weights are proportional to the capacity of the path(s) associated with egress port. Today's OpenFlow 1.x standard has limitations. The impact of these limitations on our work is described in the implementation section. To overcome the limitations of OpenFlow 1.x standard, the future generation of OpenFlow should allow the controller to tell the switch how to operate, rather than be constrained by a fixed switch design. The future switches should support flexible mechanisms for parsing packets and matching header fields [4].

This research presents SDTE, a software defined dynamic flow scheduling system which aims to balance the load and minimize the maximum link utilization (MLU) through exploiting the path diversity. SDTE exploits the path diversity through using the PEFT[1] routing algorithm for assigning non-uniform weights to the links [9]. PEFT splits the traffic along all the paths, however, it penalizes longer paths (i.e., paths with higher sums of link weights). The PEFT protocol has been proposed for wide-area ISP networks where the traffic matrix is predictable in contrast with DC traffic pattern. Thus, in this paper, PEFT was modified in order to be used in SDTE; however, the modified version has the same computational overhead as the original version. In our modified version the weights are computed and used by a central controller. In contrast to the existing TE approaches: (1) STD does not rely on the ability to predict the size of the flow, nor does it worry about the size of the new arriving flows, be it mice (short lived flows) or elephant. (2) It uses a global view of the traffic rather than a local view and knows exactly how much of the traffic must be routed from which path. (3) This system does not need to modify the end-hosts or switches.

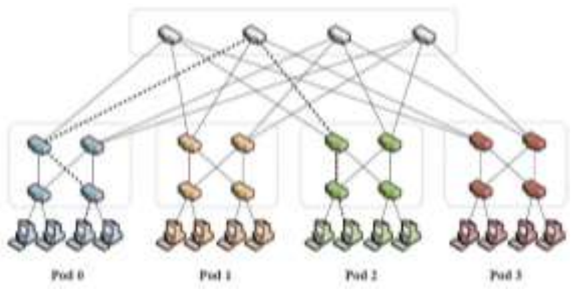

Fig. 1. Fat tree topology

In this paper, we propose an adaptive load-sensitive software defined TE approach that does not rely on the ability to predict traffic pattern. Our proposed system combines the power of software defined networking and optimality of PEFT. Additionally, we provide a practical implementation of PEFT (in java) for software defined data center networks. The rest of this paper is organized as follows. In section 2 the proposed system architecture is presented with its components and their interactions. Section 3 explains the system implementation and emulation results, and finally, section 4 concludes the paper.

II. SDTE ARCHITECTUR

In this section, the proposed architecture which is called SDTE is presented. The proposed system uses online traffic matrix (TM) calculation and assigning the non-uniform weights to links in 50s intervals. In each interval, TM will be calculated by considering all current active flows among all hosts; then, the new TM is compared with the previous time interval TM. If the difference between each entry of these two matrices is more than 20%, the weights are recalculated; otherwise, SDTE continues its work with the same weights. It means that by entrances of each flow there is no need for weight recalculation. Assigning the link weights is done through using PEFT which is an optimal routing algorithm that splits the traffic over multiple paths with an exponential penalty on longer paths that is explained in the next subsection briefly.

Figure 2 presents the components of our architecture and how they interact. SDTE consists of five components which can be classified into two groups: the executive group (EG) and the computing group (CG). The EG consists of the routing component and the path calculation component. The CG consists of the optimization component, rerouting component, and JIPOPT component. CG will compute the weights while EG use these weights for flow routing. Among the computing components, only the optimization component runs periodically. Two other computing components would also be called for weight recalculation, if needed. When a packet arrives, the routing component extracts special packet header's fields (such as source and destination MAC, IP addresses, and the network layer protocol); then sends them to the path calculation component to request a suitable path. The path calculation component replays the suitable path using this information along with the available weights. After all, the routing component installs the path on the switches. As it will be shown, the proposed architecture is quite effective in practice.

A. Overview of PEFT Routing Algorithm

PEFT is a TE mechanism that was introduced for ISP networks. Key properties of PEFT are summarized in this section. According to [9] and [10], consider a network as a directed graph G = (V, E), where V is the set of nodes (where N = |V|), E is the set of links (where E = |E|), and link (u, v) has the capacity $c_{u,v}$. The offered traffic is represented by a traffic matrix D(s, t) for source-destination pairs indexed by (s, t). TE usually considers a link-cost function $\varphi\,(\{f_{u,v}, c_{u,v}\})$ that is an increasing function of $f_{u,v}$. Since we consider the link utilization function, $f_{u,v}\,/\,c_{u,v}$, then the PEFT's TE objective is to minimize $max_{u,v\in E}\,\varphi\,(f_{u,v}, c_{u,v})$. Optimal TE requires solving the following flow conservation and link capacity constraint given by [9], whose corresponding notation is given in Table 1.

$$\min \emptyset(\{f_{u,v}, c_{u,v}\}) \qquad (1)$$

$$\text{s.t.} \sum_{v:(s,v)\in E} f^t_{s,v} - \sum_{u:(s,u)\in E} f^t_{u,s} = \mathrm{D}(s,t), \forall s \neq t$$

$$f_{u,v} = \sum_{t\in v} f^t_{u,v} \leq c_{u,v}, \forall (u,v)$$

$$\text{vars}.\, f^t_{u,v}, f_{u,v} \geq 0.$$

To offload the traffic from the congested paths to less congested but slightly longer ones, PEFT allows exponential traffic splitting over unequal-cost paths as shown in equation (2) where $p_{u,t}$ is the set of the paths from u to t and $x^i_{u,t}$ is the fraction of forwarding a

[1] Penalizing exponential flow-splitting

packet to the i-th path, i.e., $p^i_{u,t}$.

$$x^i_{u,t} = \frac{e^{-P^i_{u,t}}}{\sum_j e^{-P^j_{u,t}}} \qquad (2)$$

PEFT splits the traffic along all the available paths, but penalizes the longer paths exponentially [10].

Table 1. Key Notations

Notation	Meaning
D (s, t)	Traffic demand from source s to destination t
$c_{u,v}$	Capacity of link (u, v)
$f_{u,v}$	Flow on link (u, v)
$f^t_{u,v}$	Flow on link (u, v) destined to node t

B. *Challenges of Using PEFT Routing Algorithm on the SDTE Architecture*

PEFT is originally designed for the ISP networks. PEFT calculates the weights on an offline manner due to the predictable traffic pattern of ISP networks. In ISP networks, at first, the network operators obtain the TM from their network over long intervals and then the derived TM is used with link capacities as inputs of the link weight calculation module that exists in another independent machine. At this point, resulting links weight will be installed on switches. In contrast to the traffic pattern of the ISP networks, the measurement studies have reported that the traffic in a DC is highly dynamic and unpredictable. The lack of short term TM predictability is due to the use of random resource allocation for improving the performance of DC applications, because the distributed file system spreads the data chunks randomly across servers for load distribution and redundancy [10].

Thus, PEFT should be used in an online manner in DC networks instead of the offline manner. Previously [10] has modified PEFT for DC networks, but its applicability for software defined DC networks is not clear and has not been studied yet. In this research, the original offline PEFT was modified to be used in an online software defined TE system that is different from the original PEFT in the operational context (but it is not different in the computational context). In our implementation, the optimization component obtains the TM periodically, and when the difference between the new and old TMs is greater than 20%, the weights are recalculated in an online manner. For calculating the link weights at first, the JIPOPT component solves the optimization problem (1) and then the optimization component calculates the new weights following the functions given in [9].

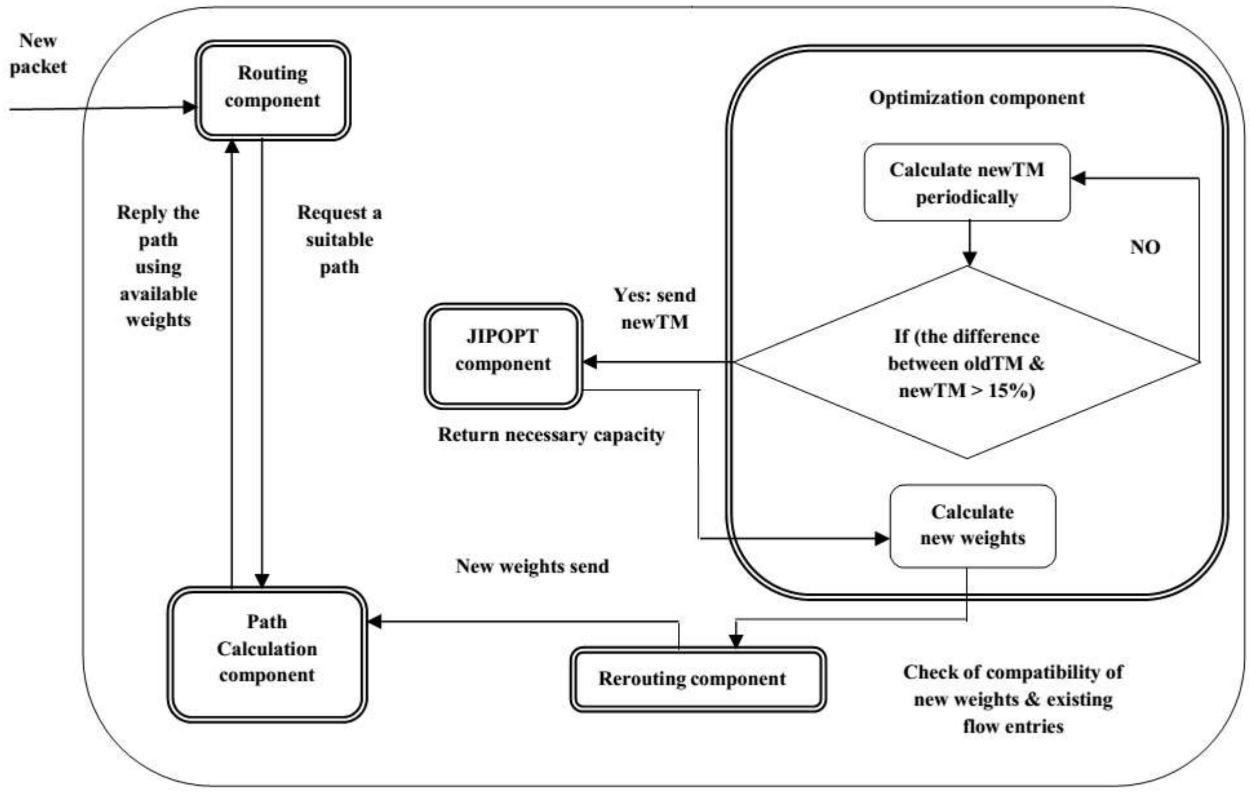
New packet
Routing component
Reply the path using available weights
Request a suitable path
Optimization component
Calculate newTM periodically
NO
Yes: send newTM
JIPOPT component
If (the difference between oldTM & newTM > 15%)
Return necessary capacity
Calculate new weights
New weights send
Path Calculation component
Rerouting component
Check of compatibility of new weights & existing flow entries

Fig. 2. SDTE architecture

C. *SDTE components*

1. Routing component

The routing component is a simple but important component that acts similar to the input component of the proposed system. When a packet that belongs to a new flow arrives, this component asks for the suitable path from the path calculation component. At this point, the routing component extracts special packet header's fields such as MAC address of source and destination, IP address of source and destination, and the network layer protocol; then it sends them to the path calculation component. The purpose of selecting these fields is to specify a unique flow. When the path calculation component replies, the routing component installs the flow table entries on all switches of the specified path. Installation of flow table entries is done by OpenFlow protocol standard messages. This component interacts with the path calculation component and requests and installs suitable paths for new flows.

2. Path calculation component

The path calculation component has a database of network topology information and uses it for path calculation of new flows. This information is provided by basic services of Beacon controller [14] which is an OpenFlow controller used for our emulation. The optimization component assigns a weight to each link of the topology due to the network load and conditions of the network congestion. The path calculation component calculates requested path based on link weights and the header information (e.g. the destination address) and sends the path to the routing component. The optimization component periodically calculates the weights for the path calculation component. In fact, the weights and consequently the paths modifies upon the changes that happen in the TM.

3. JIPOPT component

At first, this component solves the optimization problem that was mentioned in subsection A to provide the optimal distribution of the traffic (the necessary capacity[2]). Then, these necessary capacities will be sent to the optimization component. Solving the optimization problem can be done through modeling it through using AMPL (A Modeling Language for Mathematical Programming) [26] and subsequently employing an appropriate solver such as CPLEX [27] and the IPOPT solver [28]. However, this is unsuitable for our architecture, because the optimization component needs to run in an online manner. As a result, the java interface of the IPOPT's C++ library (JIPOPT) was included in our implementation. Furthermore JNI [15] was utilized to run JIPOPT.

4. Rerouting component

In each time interval, the optimization component characterize the link weights. This links weight will be used for the rest of interval. In the next time interval, if the network traffic pattern changes, the weights will be recalculated. Meanwhile, it is possible that there are some routed flows based on the previous weights which still continue in the new interval. According to the OpenFlow protocol, since the flow entries of these flows are installed on the switches flow table, they are not routed again. The controller can act in two ways: 1.The controller is aware of the flows on each link and their rate so it checks if the present path of flows is in accordance with the new weights or not. As the total link weights over all reachable paths to each destination on every switch is equal to one, therefore, if a link load is more than optimal load then another one has an under-optimal load. The controller reroutes the flows via updating the flow table entries. 2. The controller can delete all flow table entries from the source edge layer switches after weight recalculation. Both approaches increase the controller processing overhead. In the final implementation, the second approach is used because our emulation results reveal that in addition to the controller processing overhead, the first approach increases the reaction time of the controller when faced with a different traffic pattern compared to the previous interval traffic pattern.

5. Optimization component

The main component of the SDTE architecture is the optimization component that runs at every 50s interval. This component requires two pieces of global information about the network: the flow-level TM and the network topology. The first one is provided by requesting the flows statistics periodically from switches and the second one is provided by the path calculation component as discussed earlier. The functions of the optimization component are presented as follows: (1) Calculating the new (current) TM and its difference with the old one. (2) Sending the TM to the JIPOPT component, receiving the necessary capacities and calculating the new link weights if it is required. (3) Sending the new weights to the path calculation component and calling the rerouting component if needed. The pseudo code of this component is given in algorithm1. In each interval, the optimization component starts with calculating the new TM. For this purpose, this component must be able to perform the following tasks: (1) sending flow statistics request messages to switches (2) Receiving the flow statistics reply messages from switches and processing them. The information about the individual flows is requested by OFPST_FLOW stats request messages [13]. The body of replied messages includes information about the flows such as the packet count, byte count, total flow duration, and etc.

The optimization component calculates the flow-level TM through using transmitted byte count and flow duration time. After TM calculation, if the difference between the new and old TMs[3] is greater than 20%, the JIPOPT component and the optimizeOverLinkWeights() function will be called. The optimization component sends the TM to the JIPOPT component and then the JIPOPT component returns the necessary capacities values. These values are utilized by optimizeOverLinkWeights function for link weights calculation. The pseudo code of

[2]The necessary capacity is a minimal set of link capacities to realize the optimal TE [9].

[3] A comparison is made for each individual entry within the matrix.

optimizeOverLinkWeights function is presented in [9]. Due to space constraints, the extra explanations are avoided here. After these steps, the new weights is sent to the path calculation component and the rerouting component will be called. In addition to the mentioned threshold (20%), a low threshold value will be considered. The value of this threshold is 15%. Suppose the difference between a new entry and an old entry is close to 20% (but not 20%). In this case the weights will not be recalculated. But if this scenario repeats for several times, the weights must change. The low threshold will checks for this scenario.

I. IMPLEMENTATION AND EVALUATION

In this section, the performance of SDTE is evaluated with respect to improve flow rate, link utilization, and load balancing. SDTE was proposed for software defined networks therefore, a controller and a network emulator are required for its implementation. There are several SDN controllers such as Beacon [14], Nox [3], Pox [22], Floodlight [23] and OpenDayLight [24]. For our emulation, Beacon (a Java_based OpenFlow controller) and OpenVswitch (a software OpenFlow switch) running in Mininet [29] were used. Currently, the Beacon controller like most other controllers supports OpenFlow specification version 1.0. Mininet is a network emulator that runs on a single Linux kernel, which is used for building a Fat tree topology that was described in section I. We evaluated SDTE for a Fat-tree k=4 topology that contains 16 hosts, 20 switches, and 64 links. All links are 1Gb/s. In the following part, the results of the proposed system are discussed and compared with the results of random routing of flows. In the future work we will use Maxinet [30]. Maxinet extends Mininet to spans an emulated network over several physical machines, making it possible to emulate large data center networks. It also introduces a traffic generator for data center traffic.

The optimization component algorithm

```
a timer is scheduled to run every 50 seconds
flow statistics are obtained
previous trafficMatrix is stored in oldTrafficMatrix
structure
//calculation of newTrafficMatrix
foreach switch do
    send ofp_stats_request of type OFPST_FLOW //
        flow statistics request messages
    receive reply to OFPST_FLOW request
        //flow statistics response messages
    retrieval flow src & dst
    insert flow in newTrafficMatrix Entry
    store flow statistics & senderSwitch
end foreach
current trafficMatrix is stored in newTrafficMatrix structure
compare difference between oldTrafficMatrix &
newTrafficMatrix //entry by entry
if difference > 15% then
    call JIPOPT component
    call optimizeOverLinkWeights( )
    call rerouting component
end if
Continue
```

Algorithm 1: The optimization component algorithm

A. *Traffic generation*

One challenge of the proposed system performance evaluation is how to simulate and generate the cloud DC traffic pattern. At first the DC traffic patterns are investigated briefly and then the traffic generator is explained. Research articles [18, 19, and 20] have studied the traffic characterization in data center networks. Based on these studies, in cloud DCs, the majority of the traffic originated by servers (about 80%) stays within the rack. However, for other DCs such as university and private enterprise data centers, most of the traffic (40-90%) leaves the source rack for other destination racks. In the studied data centers, 80% of the flows are smaller than 10KB in size and most of the bytes are in the top 10% of large flows whose length varies from 100MB to about 1GB [10]. Also, the lifetime of 80% of the flows is less than 11 seconds. Our traffic generator uses the aforementioned characteristics and sends 20% of the traffic to external destinations or other racks while 80% of the traffic stays within the rack. The size of 20% of flows is between 500Mb and 1Gb which are considered big flows and the size of 80% of flows is between 10Kb and 1Mb which are considered small flows. In order to pressure the network, the destination of big flows is set outside of the rack and the destination of the small flows inside the rack. The lifetime of flows is 5ms.

The OpenFlow specification version that Beacon controller supports (version 1.0), limits our simulation. As mentioned before, version 1.0 does not support flow splitting. To compensate for this limitation, our traffic generator generates flows with uniform sizes. In fact, SDTE needs to split flows and without this ability, its performance would be dependent on flow sizes; therefore if the size of flows is uniform, SDTE performs close to the optimal; otherwise, it will distance from the optimal. As will be explained in the next parts, for the next version of SDTE, the flow splitting ability is added to the controller.

B. *Control messages*

SDTE uses four types of control messages that are exchanged between the controller and switches. These messages are: OFPST_FLOW messages for requesting information about flow OFPST_PORT messages for requesting information such as port's number of transmitted bytes, OFPT_FLOW_MODE messages with OFPFC_ADD command for installing new flow entries on switches flow table, and OFPT_FLOW _MODE messages with OFPFC_DELETE command for deleting flow entries from switches flow table. The first two messages are exchanged every time the optimization component runs and the next two messages are exchanged when installing a path for a new flow or deleting a flow entry from switches flow table. The Optimization component time interval is 50 seconds; however, smaller time intervals could provide a more accurate view of flows but increases the amount of control messages transmitted over the network and processed by the controller. Yet, in the next part the evaluation results show that SDTE performance is close to the optimal.

C. Flow Rate

In this section, we focus on the performance of SDTE with respect to the flow rate. First the path stretch is discussed. [10] defines the path stretch as the ratio of the length of actual path to the length of the shortest path. In our emulation which uses fat tree topology with many redundant paths, path stretch is equal to 1. This means all paths that are used for routing are shortest paths and no detour path is needed. In the following part, we used "designated rate" to refer to the bandwidth that has been designed to send the flows and "actual rate" for referring to the real bandwidth that one flow can achieve. The actual rate depends on the route that a flow takes. If this route is congested because of the load imbalance, the actual rate of flows will be reduced. If the network traffic is distributed properly, the actual rate of flows will be increased.

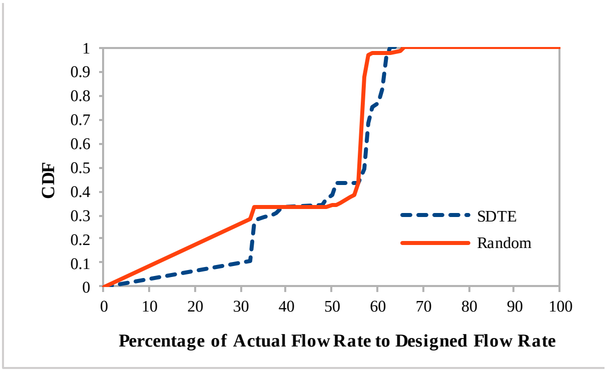

Fig.3. Comparison of actual rate of flows for SDTE and Random

For example, if the designated rate of a flow is 500MB and its actual rate is 400MB, this flow has gained 80% of its designated rate. Figure 3 shows the actual rate of flows relative to the designated rate of flows. Because the results of the emulation show that the actual rate of small flows was at least equal to their designated rate, this figure only examines the big flows. More flows can obtain a higher rate with SDTE. The number of flows with the lowest rate (about 30-40%) are equal in both SDTE and Random. SDTE provides an actual rate of 60-70% for 25% of flows while this value is around 2% for Random. Random cannot distribute the network load over all possible paths because when a new flow arrives, it routes the flow from one of the available paths without considering the network current traffic. Therefore, it is possible that two or more big flows are routed from the same path while other paths are underutilized. In Contrast to Random, SDTE does not act randomly, rather it uses unequal cost paths for the flow routing and because of this, the network load is distributed over all available paths; as a result, more flows can achieve a higher rate. The difference between SDTE and Random performance is particularly evident through investigating the link utilization. In the next part, this parameter will be analyzed.

D. Link Utilization

Figure 4a compares the link utilization of Optimal, SDTE, and Random over Fat tree topology. The Optimal values are the values that have been resulted from solving the optimization problem (1) by JIPOPT component. [9] Has proven that these values are optimal. Figure 4b compares the link utilization of Optimal and Random. In this figure, it is clear that the behavior of Random is different from Optimal. Random cannot properly distribute the load on links because it randomly selects one of the possible paths without any knowledge of current links load. Therefore, it cannot prevent the congestion over some of the links while others are idle or underutilized. It is noteworthy that in the presence of more flows and heavier traffic, Random shows a worse performance, and also in the case of more redundant paths, it hides his bad performance better. The existence of larger and smaller values compared to Optimal in figure 4b shows that there are several big flows on some of the links while the others are idle and the network load is not properly distributed i.e. the network load is not balanced. Figure 4c compares SDTE and Random.

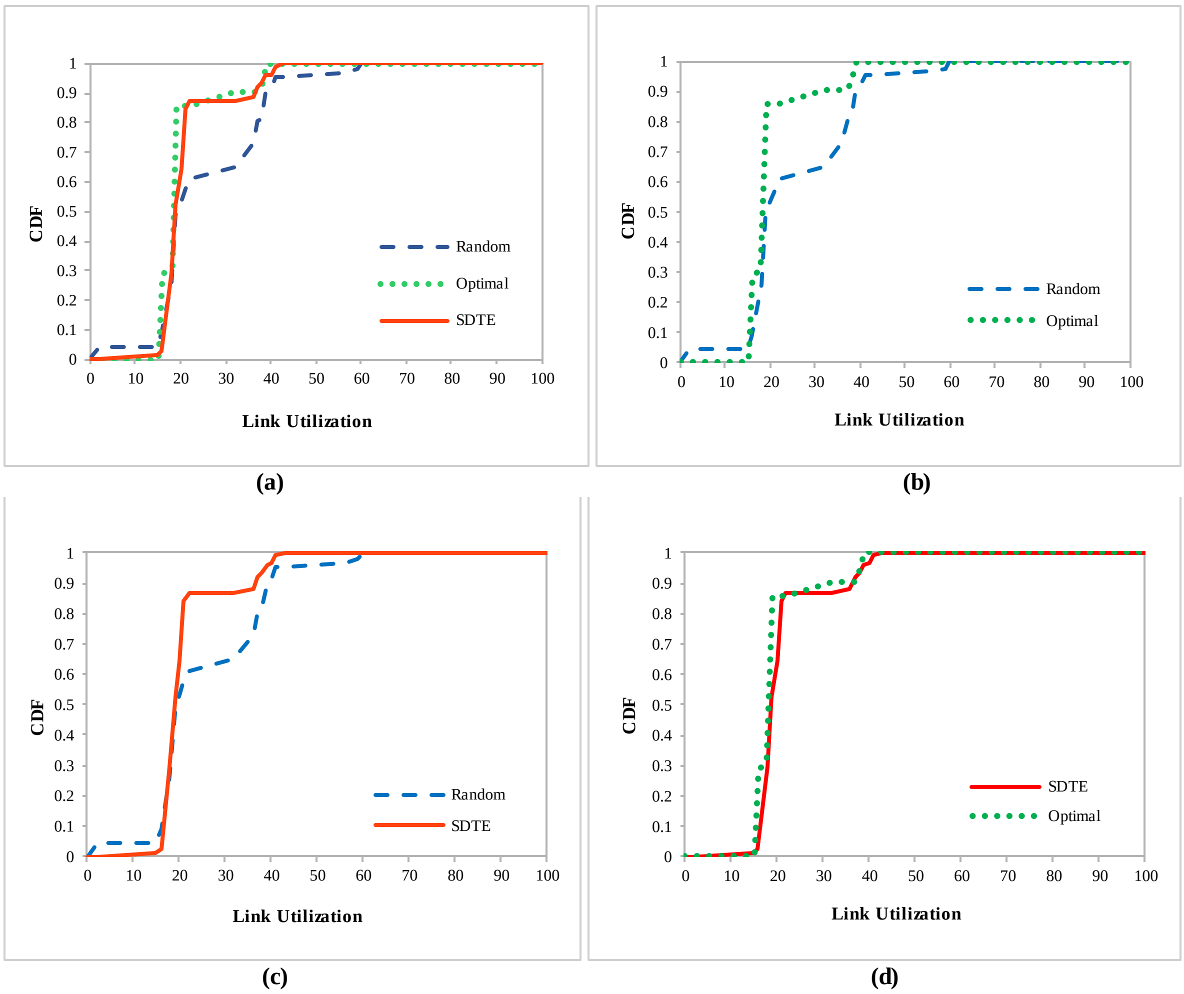

Fig. 4 . Comparison of link utilization for (a) Optimal, SDTE, and Random, (b) Random versus Optimal, (c) Random versus SDTE, (d) SDTE versus Optimal

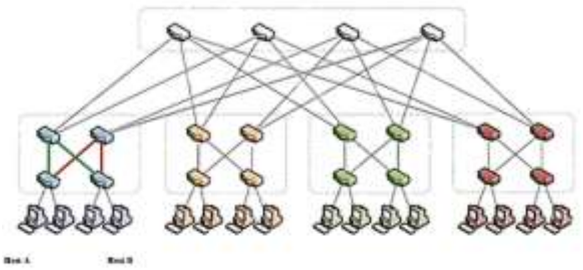

Fig.5. A flow from the source host A to the destination host B

For a better explanation, in Figure 5 there is a flow from the source host A to the destination host B with two different paths. As mentioned before, Random routes the flow from one of the possible paths without considering current load on that path or links congestion. Contrary to Random, SDTE routes the flows according to the network congestion conditions and the weights that it holds; therefore, flows are distributed on outgoing interfaces according to the optimal weights. This is the reason of the fundamental mismatch between SDTE and Random performance. Figure 4d compares link utilization of SDTE and the Optimal. The performance of SDTE is closer to the Optimal compared to Random. The question is that despite the fact of using optimal weights, why SDTE still deviates from the Optimal? As previously discussed, one reason is that the utilized controller does not support flow splitting and the other reason refers back to reactive and sparse TM calculation because of saving resources. On average, SDTE only slightly sacrifices the optimality by 7.3%, yet it provides a significant improvement over Random about 9.9%.

The maximum and minimum link utilization values of Optimal, SDTE, and Random over Fat tree topology are presented in Table 2. It can be seen that Random exhibits a wider spread in link utilization over SDTE and Optimal which implies that traffic on the links across the network is unbalanced. While the maximum link utilization of SDTE is 4% more than Optimal, this value is 17% for Random. For a closer look, the link utilization of core

Table 2. The maximum and minimum link utilization values of Optimal, SDTE, and Random

	Maximum Link Utilization	Minimum Link Utilization
Optimal	39%	16%
SDTE	43%	15%
Random	6%	2%

layer links is demonstrated in Figure 6. The worse performance of Random compared to SDTE is evident.

SDTE schedules and splits the traffic over paths to leverage the path diversity and as a result, the network traffic is balanced better, while Random shows a different behavior due to the randomness in flow scheduling. SDTE provides more links with average link utilization while Random has some links with link utilization close to zero and some with high link utilization.

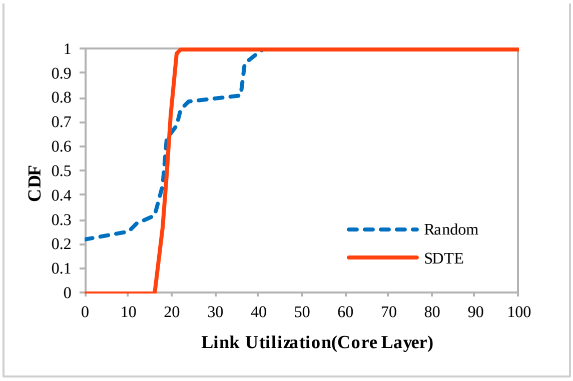

Fig. 6. Comparison of core layer link utilization for Random versus SDTE

E. System stability

Systems that are used for TE should be able to preserve their stability for different traffic patterns. It should be noted that stability is highly important for online TE systems. Each time the network traffic pattern changes, SDTE recalculates the weights. SDTE output is an appropriate path for flows based on these weights. If the recalculation of weights is done repeatedly, the system loses its stability and causes route oscillation. The proposed system can prevent this instability through assigning a suitable time interval (this time interval is 50s in our emulation) for the optimization component to recalculate weights when facing different traffic patterns. However, this interval causes SDTE to deviate from Optimal, yet is effective and providing a performance near to Optimal.

F. System overhead

system overhead can be studied from two aspects, 1) time complexity of optimization component and 2) the number of input flows to routing component. optimization component uses optimize over link weights function. This function requires Floyd-Warshal algorithm to compute the all-pairs shortest paths that has time complexity $O(N^3)$. Also, this function requires topology sorting. For each destination, topology sorting needs O(N+E) time, and summarizing the incoming flow and splitting across the outgoing links requires O(N+E). Thus, the total time complexity to calculate the traffic distribution is $O\big(N^3 + N(N+E)\big) = O(N^3)$ [9].

In current version of SDTE, switches inform routing component with each new flow. Then routing component specify route to the new flow. It increases overhead of the controller beacuse the median arrival rate of all flows is 10^5 flows per second in DCNs [19]. In future, we will use group table [21] to decrease overhead of the controller. With group table, some decisions are made locally by switches. Therefore, no need to send all new flows to the controller.

II. CONCLUSION AND FUTURE WORK

In this paper, we proposed SDTE (an online load sensitive software defined traffic engineering system) to improve link utilization, load balancing and performance over cloud data center networks. SDTE distributes network traffics among all available paths. SDTE uses PEFT which assigns a weight for each link/destination to achieve optimal traffic distribution. We modified PEFT to make it compatible with both DC traffic pattern fluctuations and OpenFlow framework. The evaluation shows that the proposed system results are close to the optimal solution. Therefore, it can distribute the load on all available paths, this leads to reduction of maximum link utilization. Moreover, the evaluation confirmed that SDTE improves the rate of flows. Due to limitations of current implemented version of OpenFlow (version1.0) the proposed system does not support flow splitting. Future works would be dedicated to extend SDTE to support flow splitting and group table (a new feature in OpenFlow version1.1 and later versions through which packets are processed based on a switch-computed selection algorithm).

References

[1] S. Vissicchio, L. Vanbever and J. Rexford, "Sweet Little Lies: Fake Topologies for Flexible Routing", *Proceedings of the 13th ACM Workshop on Hot Topics in Networks - HotNets-XIII*, 2014.

[2] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers", *Proceedings of the Ninth European Conference on Computer Systems - EuroSys '14*, 2014.

[3] 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 Communication Review*, vol. 38, no. 3, p. 105, 2008.

[4] P. Bosshart, G. Varghese, D. Walker, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco and A. Vahdat, "P4: Programming Protocol-Independent Packet Processors", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87-95, 2014.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks", *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pp. 19-19, 2010.

[6] A. Curtis, W. Kim and P. Yalagandula, "Mahout: Low-Overhead Datacenter Traffic Management Using End-Host-Based Elephant Detection", *2011 Proceedings IEEE INFOCOM*, 2011.

[7] T. Benson, A. Anand, A. Akella and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers", *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies on - CoNEXT '11*, 2011.

[8] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, 2002.

[9] Dahai Xu, Mung Chiang and J. Rexford, "Link-State Routing With Hop-by-Hop Forwarding Can Achieve Optimal Traffic Engineering", *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1717-1730, 2011.

[10] F. Tso and D. Pezaros, "Improving Data Center Network Utilization Using Near-Optimal Traffic Engineering", *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1139-1148, 2013.

[11] M. Al-Fares, A. Loukissas and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, p. 63, 2008.

[12] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network", *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, p. 51, 2009.

[13] "OpenFlow switch specification", Open Networking Foundation, ver. 1.0, 2009.

[14] Openflow.stanford.edu, "Home - Beacon - Confluence". Available: https://openflow.stanford.edu/display/Beacon/Home.

[15] S. Liang, *The Java Native Interface*. Reading, Mass.: Addison-Wesley, 1999.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008.

[17] R. Bohn, J. Messina, F. Liu, J. Tong and J. Mao, "NIST Cloud Computing Reference Architecture", *2011 IEEE World Congress on Services*, 2011.

[18] T. Benson, A. Anand, A. Akella and M. Zhang, "Understanding Data Center Traffic Characteristics", *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, p. 92, 2010.

[19] S. Kandula, S. Sengupta, A. Greenberg, P. Patel and R. Chaiken, "The Nature of Data Center Traffic: Measurements &Analysis", *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09*, 2009.

[20] T. Benson, A. Akella and D. Maltz, "Network Traffic Characteristics of Data Centers in the Wild", *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, 2010.

[21] "OpenFlow switch specification", Open Networking Foundation, ver. 1.4, 2013.

[22] Openflow.stanford.edu, "POX Wiki - Open Networking Lab - Confluence". Available: https://openflow.stanford.edu/display/ONL/POX+Wiki.

[23] Project Floodlight, "Floodlight OpenFlow Controller -". Available: http://www.projectfloodlight.org/floodlight/.

[24] J. Medved, R. Varga, A. Tkacik and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller Architecture", *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[25] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP", *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 29-29, 2012.

[26] R. Fourer, D. Gay and B. Kernighan, *AMPL, a Modeling Language for Mathematical Programming*. San Francisco, CA: Scientific Press, 1993.

[27] Www-01.ibm.com, "IBM CPLEX Optimizer - United States". Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[28] A. Wächter and L. Biegler, "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming", *Mathematical Programming*, vol. 106, no. 1, pp. 25-57, 2005.

[29] B. Lantz, B. Heller and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks", *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, 2010.

[30] P. Wette, M. Draxler and A. Schwabe, "MaxiNet: Distributed Emulation of Software-Defined Networks", *2014 IFIP Networking Conference*, 2014.

[31] D. Wischik, C. Raiciu, A. Greenhalgh and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP", *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pp. 99-112, 2011.

[32] N. Kang, M. Ghobadi, J. Reumann, A. Shraer and J. Rexford, "Efficient Traffic Splitting on Commodity Switches", 2015.

[33] P. Sun, L. Vanbever and J. Rexford, "Scalable Programmable Inbound Traffic Engineering", *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15*, 2015.

[34] S. Vissicchio, O. Tilmans, L. Vanbever and J. Rexford, "Central Control Over Distributed Routing", *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 43-56, 2015.

Marzieh Khoshbakht received her B.Sc. degree in computer engineering from Bonab PNU University, East Azarbaijan, in 2010. She received her M.Sc. degree in computer engineering from Tarbiat Modares University, Tehran, in 2013. Her main research interests include software defined networking, traffic engineering, and cloud computing.

Mohammad Mahdi Tajiki received his B.Sc. degree in Computer Engineering from Shahid Bahonar University, Kerman, Iran, in 2011. In 2013, he graduated from Electrical and Computer Engineering School of Tehran University, Tehran, Iran. Currently, he is a Ph.D. candidate in Tarbiat Modares University, Tehran, Iran. His main research interests are network QoS, media streaming over the Internet, data center networking, traffic engineering, and software defined networking (SDN).

Behzad Akbari received his B.Sc., M.Sc. and Ph.D degree in Computer Engineering from Sharif University of Technology, Tehran, Iran, in 1999, 2002 and 2007 as an assistant professor, respectively. He joined, to department of Electrical and Computer Engineering at Tarbiat Modares University, Tehran, Iran in 2007. His main research interests are Internet QoS, media streaming over the Internet, peer-to-peer networks, peer-to-peer video streaming, wireless video communications, data center networking, software defined networking (SDN) and network performance modeling and evaluation.