On the Effect of Flow Table Size and Controller Capacity on SDN Network Throughput

Gongming Zhao, Liusheng Huang, Zhuolong Yu, Hongli Xu, Pengzhan Wang
School of Computer Science and Technology, University of Science and Technology of China, China
{zgm1993, yzl123, pzwang}@mail.ustc.edu.cn, {lshuang, xuhongli}@ustc.edu.cn

***Abstract*—Software Defined Network (SDN) is an architectural trend in networking towards the use of the centralized controller to get better performance. However, due to limited resources (especially limited flow table size and controller processing capacity), it may result in low-throughput and long-delay for a set of bursty flows. In this paper, we first combine the flow table size constraint and the controller processing capacity constraint to define the Throughput Maximization with Limited Resources (TMLR) problem. Then we prove TMLR is NP-Hard and design an approximation algorithm to solve the TMLR problem. The approximation factor of the proposed algorithm is also analyzed. The simulation results on the SDN platform (Mininet [1]) show that our algorithm can improve the network throughput about 39% on average compared with the existing algorithms.**

Index Terms*—SDN, Throughput-Maximization, Flow Table Size Constraint, Controller Capacity Constraint.

I. Introduction

Software Defined Network (SDN) provides a new centralized architecture with flexible network resource management to support a huge amount of data transmission. Different from legacy networks, SDN separates the control plane from switches and allows the control plane to be programmable to efficiently optimize the network resources.

However, when dealing with bursty flows (a large number of flows arrived in a short time slot, which is normal during peak periods), the SDN will encounter two main capacity constraints as follows, resulting in unacceptable performance (*e.g.*, low-throughput, long-delay, high packet-loss ratio).

Flow table size constraint. SDN switch usually contains less than $5K$ flow entries [2] because of the high price and energy-consuming of TCAM. However, for a content distribution network, a service provider network or even a large campus network, the number of concurrent flows through a key switch can easily exceed tens of thousands, overwhelming its flow table. For example, in a moderate-size datacenter network, the average arrival rate will reach $22K$ flows per second for a rack consisting 40 servers hosting around 15 virtual machines per host [3].

Controller processing capacity constraint. Switches will ask the controller for forwarding rules using PacketIn interface [4] when new flows arrivals. The controller needs to send the packet-out messages [4] for all edge switches (for ARP) and the flow-mod messages for switches on the path (for installing rules). Processing time for sending the flow-mod message and packet-out message are about $0.5ms$ and $0.2ms$, respectively [5]. Suppose that the average flow-path goes through 5 switches and this network has 10 edge switches, the controller will spend $4.5ms$ for only one flow! Therefore, when the new bursty flows arrive, we need to find a solution to reduce the controller's load (or deployment delay).

Some works have dedicated to solving these constraints. Most of these works only concentrated on one of the constraints. The authors of [6] focused on the limited TCAM size by designing new placement rules. The authors of [7] devoted to solving the shortage of the controller capacity by using multi-controllers. However, the two constraints are both fatal when bursty flows arrive. On the one hand, if flow entries are insufficient, some flows may have to be dropped (it decreases network throughput) or take turns using flow entries (it further increases the controller's load). On the other hand, if the communications between the controller and the switches exceed the controller's capacity, the network will become unstable (cause high packet-loss ratio) and the controller's processing time will increase (cause high packet-delay).

In this paper, we first define the Throughput Maximization with Limited Resources (TMLR) problem, then propose an approximation algorithm to solve the TMLR problem and analyze the approximate performance. Our objective is to get maximum throughput with limited resources (*e.g.*, TCAM resources, controller capacity and link capacity). By combining the per-flow routing and OSPF routing, and considering both flow table size constraint and controller processing capacity constraint, our proposed method achieves better performance.

II. Related Works

SDN is introduced to improve the network performance with the centralized control mechanism. However, limited resources restrict the advantages of centralized control services, especially handling bursty flows. It is important to solve these constraints for throughput maximization in an SDN. We review the related works that dealing with the two main critical constraints, including flow table size constraint and controller processing capacity constraint in this section.

Solving flow table size constraint. Since Ternary Content Addressable Memory (TCAM) is expensive and energy intensive, SDN switch usually contains less than $5K$ flow entries [2]. Previous works mainly use the wildcard-based methods for traffic aggregation or only route elephant flows to reducing the use of flow entries. Deveflow [2] proposed the sampling and trigger techniques to figure out elephant flows, then deployed flow entries only for these elephant flows. Similar works also have been proposed by AL-Fares et al. [8]. The authors of [9] formulated the throughput maximization problem and proposed approximation algorithm. This work simply dropped flows that can not fit in the flow entries. ISTAMP [10] used part of the TCAM rules for aggregation traffic, but it may encounter the aggregation feasibility problem in the network as not all the flows can be aggregated. Both two works [11] [12] used wildcards in the flow entries so that each entry matches multiple flows. Besides, the authors of [6] focused on the limited TCAM rules, by designing new fabric.

Solving controller processing capacity constraint. Due to the restricted capacity of the controller, some works devoted to improving controller capacity. The authors of [7] dedicated to solving the shortage of the controller capacity by using multi-controllers. Work [13] proposed an elastic distributed controller architecture to improve the performance of multi-distributed controllers. The authors of [14] concentrated on multiple controllers by dynamically adjusts the number of active controllers and delegates each controller with a subset of OpenFlow switches according to network dynamics. The authors of [15] studied the dynamic controller assignment (DCA) problem as a stable matching problem with transfers to minimize the controller response time.

All the above works have studied the limited resources problem for SDNs, so as to obtain throughput maximization. However, there are some disadvantages. First, all the works about wildcards could lead to low-throughput. Because the wildcards (such as prefix, suffix) is constant and we can not choose the optimal path for per-flow routing. It would cause routing inflexibility and per-flow QoS difficulty. Second, the works choosing a part of flows to route also lead a low QoS. They dropped the unimportant flows if the requested resources are not met. Third, Using multiple controllers can only solve the controller processing capacity constraint. Besides, it is difficult for management. Fourth, most of these works only focused on one of the constraints.

In this paper, We propose an approximation algorithm to obtain throughput maximization with limited resources (including controller constraint, flow table constraint and link constraint) in SDNs, especially for bursty flows.

III. Preliminaries and Problem Formulation

In this section, we introduce a network model for SDNs and define the Throughput Maximization with Limited Resources (TMLR) problem.

A. Network Model

An SDN typically consists of three device sets: an SDN controller c, an SDN switch set $S = \{s_1, ..., s_{|S|}\}$; and a host set $H = \{h_1, ..., h_{|H|}\}$. These switches and hosts comprise the forwarding plane of an SDN. The controller is responsible for route selection. In other words, it will not participate into packet forwarding. Then, the network topology can be modeled by $G = (H \cup S, E)$, where E is a set of links. Note that, any two terminals will be connected only through one or several switches. For ease of expression, let $c(e)$ and $n(s)$ denote the capacity of a link $e \in E$ and the number of flow entries of a switch $s \in S$, respectively. Besides, let $c(c)$ denotes the capacity of the controller. The main task of the controller is monitoring network and sending Flow Entry Installation Messages (flow-mod commands [4]). The required resources $c_m(c)$ for monitoring network is usually constant and available, thus, the capacity of sending Flow Entry Installation Messages is nearly constant. For example, the normal controller can reply flow-mod messages at the speed of 2K/Second [5].

B. Definition of Throughput Maximization with Limited Resources (TMLR)

When a set of bursty flows Γ arrives, we need to deploy flow entries for these flows $\Gamma = \{\gamma_1, ..., \gamma_{|\Gamma|}\}$, in the network. Each flow may contain the request for some designated resources, thus it is assumed that the controller can get the traffic demand $f(\gamma)$ for each $\gamma \in \Gamma$. For example, when a terminal uses the high-definition video conference, its traffic demand is 4Mbps. At the same time, the standard-definition video conference takes a bandwidth of 2Mbps. We use $\mathcal{P}_\gamma$ represents the several feasible paths from source to destination for each flow $\gamma \in \Gamma$. These feasible paths can be got by Weighted Dijkstra methods [16]. Besides, each flow γ can be forwarded by traditional OSPF path and the OSPF path $d(\gamma) \in \mathcal{P}_\gamma$. One OSPF path can be shared by many flows and is a widely used default routing. Note that, the OSPF paths can be pre-deployed and not consume the controller processing capacity when the new bursty flows arrive. We use $\mathcal{P}'_\gamma$ represents per-flow based paths set, which equals the paths set $\mathcal{P}_\gamma - \{d(\gamma)\}$. Our objective is to assign the bursty flows and get the maximum throughput.

By using OpenFlow Interfaces [4], we can estimate the already used link capacity $c_u(e)$ for each link e and the already used number of flow entries $n_u(s)$ for each switch s. We formulate the TMLR problem into a non-linear program as follows. The variable z_γ denotes whether the flow γ can be served by an SDN or not. Let variable $y_\gamma^p \in \{0, 1\}$ denote whether the flow γ selects the feasible path $p \in \mathcal{P}_\gamma$ or not. Note that, we assume each flow is unsplittable to save flow entries and group entries. TMLR solves the following problem:

$$\max \quad \sum_{\gamma\in\Gamma} z_\gamma \cdot f(\gamma)$$

$$S.t. \begin{cases} z_\gamma = \sum_{p\in\mathcal{P}_\gamma} y_\gamma^p \le 1, & \forall\gamma \in \Gamma \\ \sum_{\gamma\in\Gamma}\sum_{s\in p: p\in\mathcal{P}'_\gamma} y_\gamma^p \le n(s) - n_u(s), & \forall s \in S \\ \sum_{\gamma\in\Gamma}\sum_{e\in p: p\in\mathcal{P}_\gamma} y_\gamma^p f(\gamma) \le c(e) - c_u(e), & \forall e \in E \\ \sum_{s\in S}\sum_{\gamma\in\Gamma}\sum_{s\in p: p\in\mathcal{P}'_\gamma} y_\gamma^p \le \phi(c(c) - c_m(c)), & \\ y_\gamma^p \in \{0, 1\}, & \forall p, \gamma \end{cases} \tag{1}$$

The first set of inequalities means that each flow will be assigned a feasible path from source to destination at most. The second set of inequalities denotes that the required flow entries on each switch s should not exceed the rest of the flow entries. The third set of inequalities expresses that the sum of traffic amounts from all the bursty flows after route selection on each link e, does not exceed the link's remaining capacity, so that the network congestion can be avoided. The fourth set of inequalities implies that the task of sending Flow Entry Installation Messages (flow-mod commands) should not exceed the controller's remaining capacity. ϕ is a converting factor between the controller capacity and the flow command number. Recent studies show that delay is critical for many data center systems, *e.g.*, 100ms delay causes a 1% drop in revenue at Amazon and 400ms delay causes a 5-9% decrease in traffic at Google [17]. Thus, we can use the expected deployment delay to denote the controller processing capacity for simplicity. For example, if the controller can send flow-mod commands at the speed of $2K$/Second [5] and we want these bursty flows can be forwarded in 4 seconds. That means, the $\phi \cdot (c(c) - c_m(c)) = 8K$. Our objective is to maximize the network throughput. That is, $\max \sum_{\gamma \in \Gamma} z_\gamma \cdot f(\gamma)$.

Theorem 1: The TMLR problem defined in Eq. (1) is an NP-hard problem.

Proof: We consider a special example of the TMLR problem, in which there is no constraints on controller processing capacity and flow table size. Then, we are able to deploy rules for all the flows in an SDN so as to achieve throughput maximization with link capacity constraints. In other words, this becomes an unsplittable multi-commodity flow with maximum throughput problem [18], which is NP-hard. Since the multi-commodity flow problem is a special case of our problem, the TMLR problem is NP-hard too. ■

IV. Algorithm Description for the TMLR Problem

In this section, we first present an approximation algorithm (Randomized Rounding-based Throughput Maximization, RRTM) to solve the TMLR problem, and then analyze the approximation performance of the proposed algorithm.

A. Approximation Algorithm to Solve TMLR

Due to NP-hardness, this section presents an approximation algorithm to deal with the TMLR problem. We design the Randomized Rounding-based Throughput Maximization (R-RTM) algorithm to solve the TMLR problem which is NP-hard. To solve this problem in polynomial time, we relax this assumption to assume that each flow can be split and forwarded through multiple paths. By relaxing this assumption, y_γ^p is fractional. So we can solve it in polynomial time with a linear program solver, such as CPLEX [19]. Assume that the optimal solution is denoted by $\widetilde{y}$, and the optimal result is denoted by $\widetilde{F}(\widetilde{F} = \sum_{\gamma \in \Gamma} z_r \cdot f(\gamma))$. Obviously, $\widetilde{F}$ is an upper-bound result for TMLR. Using the randomized rounding method [20], we obtain an integer solution $\widehat{y}_\gamma^p$. More specifically, for each flow $\gamma \in \Gamma$, we select a feasible path $p \in \mathcal{P}_\gamma$ with the probability of $\widetilde{y}_\gamma^p$ for flow γ. If $\exists p \in \mathcal{P}_\gamma, \widehat{y}_\gamma^p = 1$, it means that flow γ selects $p \in \mathcal{P}_\gamma$ as its finally route path. If $\forall p \in \mathcal{P}_\gamma, \widehat{y}_\gamma^p = 0$, it means that flow γ will not be served. By this way, we have determined the finally route paths for all the flows in an SDN. The RRTM algorithm is formally described in Alg. 1.

Algorithm 1 RRTM: Randomized Rounding-based Throughput Maximization

```
1:  **Step 1: Solving the Relaxed TMLR Problem**
2:  Explore a feasible path set $\mathcal{P}_\gamma$ for each flow $\gamma \in \Gamma$
3:  Construct a linear program $LP_1$ based on Eq.(1)
4:  Obtain the optional solution $\widetilde{y}$
5:  **Step 2: Route Selection for Throughput Maximization**
6:  Derive an integer solution $\widehat{y}_\gamma^p$ by randomized rounding
7:  for each flow $\gamma \in \Gamma$ do
8:      for each routh path $p \in \mathcal{P}_\gamma$ do
9:          if $\widehat{y}_\gamma^p = 1$ then
10:             Appoint a feasible path $p$ for flow $\gamma$
11:         end if
12:     end for
13: end for
```

B. Approximate Performance Analysis

This part will analyze the approximate performance of the proposed RRTM algorithm. More specifically, we prove that the total traffic on any link $e \in E$ would not exceed the traffic of the fraction solution by a factor β and the total flow entries on any switch $h \in H$ would not exceed the time of the fraction solution by a factor δ. Besides, the required controller processing capacity would not exceed the time of the fraction solution by a factor λ. That means, we will calculate factors β, δ and λ to prove the proposed algorithm have a fine performance. We first give two famous theorems for probability analysis.

Theorem 2 (Chernoff Bound): Given n independent variables: $x_1, x_2, ..., x_n$, where $\forall x_i \in [0, 1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^n x_i]$. Then, $\mathbf{Pr}\left[\sum_{i=1}^n x_i \geq (1+\epsilon)\mu\right] \leq e^{\frac{-\epsilon^2 \mu}{2+\epsilon}}$, where ϵ is an arbitrarily positive value.

Theorem 3 (Union Bound): Given a countable set of n events: $A_1, A_2, ..., A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup ... \cup A_n) \leq \sum_{i=1}^n \mathbf{Pr}(A_i)$.

Link Capacity Constraint. We first define variable β', it means the bursty traffic ($\sum_{\gamma \in \Gamma} \sum_{e \in p: p \in \mathcal{P}_\gamma} \widehat{y}_\gamma^p f(\gamma)$) on any link $e \in E$ would not exceed the rest capacity ($c(e) - c_u(e)$) by a factor β'. Thus, we get:

$$\begin{aligned}
&\frac{\sum_{\gamma \in \Gamma} \sum_{e \in p: p \in \mathcal{P}_\gamma} \widehat{y}_\gamma^p f(\gamma) + c_u(e)}{c(e)} \\
&= \frac{\sum_{\gamma \in \Gamma} \sum_{e \in p: p \in \mathcal{P}_\gamma} \widehat{y}_\gamma^p f(\gamma)}{c(e) - c_u(e)}(1 - \frac{c_u(e)}{c(e)}) + \frac{c_u(e)}{c(e)} \\
&\Rightarrow \beta = \beta'(1 - \frac{c_u(e)}{c(e)}) + \frac{c_u(e)}{c(e)} = \beta' - (\beta' - 1)\frac{c_u(e)}{c(e)} \qquad (2)
\end{aligned}$$

Then we bound the probability with which the link capacities will be violated. Note that, after the linear program procedure in the RRTM algorithm, we derive a fractional solution $\widetilde{y}_\gamma^p$ for the relaxed TMLR problem. Thus, the traffic load of link e from flow γ could be defined as a random variable $x_{\gamma,e}$, which equals $f(\gamma)$ with probability $\sum_{p\ni e:p\in\mathcal{P}_\gamma}\widetilde{y}_\gamma^p$, or 0 otherwise, as defined in Eq. (3).

Definition 1: For each $\gamma\in\Gamma$ and each $e\in E$, a random variable $x_{\gamma,e}$ is defined as:

$$x_{\gamma,e}=\begin{cases} f(\gamma), \text{ with probability } \sum_{p\ni e:p\in\mathcal{P}_\gamma}\widetilde{y}_\gamma^p \\ 0, \text{ otherwise}\end{cases} \tag{3}$$

On the one hand, based on the definition of $x_{\gamma,e}$ and β', when network size grows, we know:

$$\mathbf{Pr}\left[\bigvee_{e\in E}\sum_{\gamma\in\Gamma}\frac{x_{\gamma,e}}{c(e)-c_u(e)}\geq\beta'\right]\rightarrow 0 \tag{4}$$

By applying Theorem 3:

$$\mathbf{Pr}\left[\bigvee_{e\in E}\sum_{\gamma\in\Gamma}\frac{x_{\gamma,e}}{c(e)-c_u(e))}\geq\beta'\right]$$
$$\leq |E|\cdot\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{\gamma,e}}{c(e)-c_u(e)}\geq\beta'\right] \tag{5}$$

Note that, the number of links $|E|$ should be not more than n^2 (n means the number of switches). Combining Eq. (4) and Eq. (5), we can know there exists $\mathcal{F}$ satisfying:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{\gamma,e}}{c(e)-c_u(e)}\geq\beta'\right]\leq\frac{\mathcal{F}}{n^2} \tag{6}$$

where $\mathcal{F}$ is a function of network-related variables (such as the number of switches n, the number of links $|E|$, *etc.*). Note that, $\mathcal{F}\rightarrow 0$ when the network size grows.

On the other hand, according to this definition, $x_{\gamma_1,e}$, $x_{\gamma_2,e}$,... are mutually independent. The expected traffic load on link e is:

$$\mathbb{E}\left[\sum_{\gamma\in\Gamma}x_{\gamma,e}\right]=\sum_{\gamma\in\Gamma}\mathbb{E}\left[x_{\gamma,e}\right]\leq c(e)-c_u(e) \tag{7}$$

By Eq. (7), we have the following conclusions:

$$\frac{x_{\gamma,e}}{c(e)-c_u(e)}\in[0,1],\quad \mathbb{E}\left[\sum_{\gamma\in\Gamma}\frac{x_{\gamma,e}}{c(e)-c_u(e)}\right]\leq 1 \tag{8}$$

We explore a variable α as follows:

$$\alpha=\min\{\frac{(c(e)-c_u(e))_{\min}}{f(\gamma)},(n(s)-n_u(s))_{\min},$$
$$\frac{c(c)-c_m(c)}{(\sum_{\gamma\in\Gamma}\sum_{p\ni s:p\in\mathcal{P}_\gamma}y_\gamma^p)_{\max}},\ \forall e\in E,s\in S,\gamma\in\Gamma\} \tag{9}$$

Combining Eq. (8) and the definition of α, we have:

$$\frac{\alpha\cdot x_{\gamma,e}}{c(e)-c_u(e)}\in[0,1],\qquad \mathbb{E}\left[\sum_{\gamma\in\Gamma}\frac{\alpha\cdot x_{\gamma,e}}{c(e)-c_u(e)}\right]\leq\alpha \tag{10}$$

Then, we set β' = 1+ρ (obviously, ρ is a positive variable), combining Theorem 2 and Eq. (6), we have:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{\gamma,e}}{c(e)-c_u(e)}\geq(1+\rho)\right]\leq e^{\frac{-\rho^2\alpha}{2+\rho}}\leq\frac{\mathcal{F}}{n^2} \tag{11}$$

Thus, we get:

$$\rho\geq\frac{2\log\frac{n^2}{\mathcal{F}}}{2\alpha}+2,\quad n\geq 2 \tag{12}$$

We set $\mathcal{F}=\frac{1}{n^2}$, then Eq. (12) can be transformed into:

$$\beta'=1+\rho\geq 1+\frac{2\log n^4}{2\alpha}+2=\frac{4\log n}{\alpha}+3,\quad n\geq 2 \tag{13}$$

According to Eq. (2), we get:

$$\beta=\frac{4\log n}{\alpha}+3-(\frac{4\log n}{\alpha}+2)\frac{c_u(e)}{c(e)},\quad n\geq 2 \tag{14}$$

Flow Table Size Constraint. Similar to link capacity constraint, we define random variable δ', $n_{\gamma,s}$ and $\mathcal{F}$. In addition, set σ +1 = δ'. Similar to Eqs. (6), (7), (10) and (11), we have:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{n_{\gamma,s}}{n(s)-n_u(s)}\geq(1+\sigma)\right]\leq e^{\frac{-\sigma^2\alpha}{2+\sigma}}\leq\frac{\mathcal{F}}{n} \tag{15}$$

Similar to Eqs. (12) and (13):

$$\sigma\geq\frac{\log\frac{n}{\mathcal{F}}+\sqrt{\log^2\frac{n}{\mathcal{F}}+8\alpha\log\frac{n}{\mathcal{F}}}}{2\alpha}$$
$$\Rightarrow\delta'=\frac{3\log n}{\alpha}+3,\quad n\geq 2 \tag{16}$$

Similar to Eq. (14):

$$\delta=\frac{3\log n}{\alpha}+3-(\frac{3\log n}{\alpha}+2)\frac{n_u(s)}{n(s)},\quad n\geq 2 \tag{17}$$

Controller Processing Capacity Constraint. Similar to flow table size constraint, we define random variable λ' and the same variable $n_{\gamma,s}$, $\mathcal{F}$. Besides, set ω +1 = λ'. Similar to Eqs. (6), (7), (10) and (11):

$$\mathbf{Pr}\left[\sum_{s\in S}\sum_{\gamma\in\Gamma}\frac{n_{\gamma,s}}{n(s)-n_u(s)}\geq(1+\omega)\right]\leq e^{\frac{-\omega^2\alpha}{2+\omega}}\leq\mathcal{F} \tag{18}$$

Similar to Eqs. (12) and (13):

$$\omega\geq\frac{\log\mathcal{F}+\sqrt{\log^2\mathcal{F}+8\alpha\log\mathcal{F}}}{2\alpha}$$
$$\Rightarrow\lambda'=\frac{2\log n}{\alpha}+3,\quad n\geq 2 \tag{19}$$

Similar to Eq. (14):

$$\lambda=\frac{2\log n}{\alpha}+3-(\frac{2\log n}{\alpha}+2)\frac{c_m(c)}{c(c)},\quad n\geq 2 \tag{20}$$

Approximation Ratio. With these analyses, we know the approximation factors for the link capacity constraint, flow table size constraint and controller processing capacity constraint are $\frac{4\log n}{\alpha}+3-(\frac{4\log n}{\alpha}+2)\frac{c_u(e)}{c(e)}$, $\frac{3\log n}{\alpha}+3-(\frac{3\log n}{\alpha}+2)\frac{n_u(s)}{n(s)}$ and $\frac{2\log n}{\alpha}+3-(\frac{2\log n}{\alpha}+2)\frac{c_m(c)}{c(c)}$, respectively. For example, let $\alpha=50, n=1000$ and $\frac{c_m(c)}{c(c)}=\frac{n_u(s)}{n(s)}=\frac{c_m(c)}{c(c)}=\frac{2}{3}$. According to the definition of these variables, this assumption is reasonable. Then the approximation factors for link capacity constraint, flow table size constraint and controller processing capacity constraint are 1.75, 1.73 and 1.72, respectively.

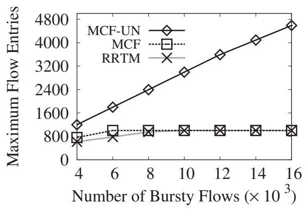

Fig. 1: Maximum Number of Flow Entries vs. Number of Brusty Flows for Topology (a)

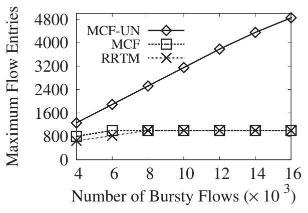

Fig. 2: Maximum Number of Flow Entries vs. Number of Brusty Flows for Topology (b)

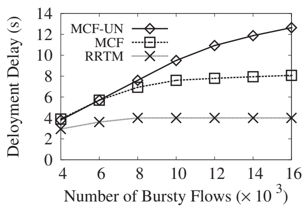

Fig. 3: Deployment Delay vs. Number of Brusty Flows for Topology (a)

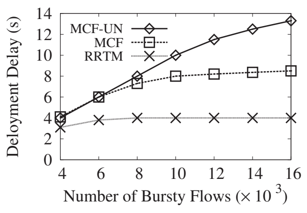

Fig. 4: Deployment Delay vs. Number of Brusty Flows for Topology (b)

V. Performance Evaluation

In this section, we first introduce the simulation settings and performance metrics, then compare our methods with the previous methods by running extensive simulations. Note that, we choose the OpenrDayLight [21] Lithium-SR1 release as the controller running on a server with a core i5-3470 processor and 8GB of RAM, and execute simulations on the Mininet platform [1], which is widely-used for the SDN.

A. Performance Metrics

To evaluate the performance of our algorithm, we compared them with serval other reference methods. One is the multi-commodity flow (MCF) method for unsplittable flows in an SDN using randomized rounding [20]. Similar MCF method is also used by work [9]. Since one flow entry will be matched with only one flow, the flow table size constraint may likely be violated by MCF. To be practical, the controller should drop some flows so as to satisfy the flow table size constraint. We use MCF-UN to denote that MCF is applied in an SDN with an unlimited flow table size. The other is the optimal result for the linear program LP_1 after relaxing the Eq. (1), denoted by OPT. Since $LP1$ is the relaxed version of the TMLR problem, OPT is an upper-bound for RRTM.

We employ three metrics for performance evaluation. When a set of bursty flows arrives, the controller will deploy paths for these flows, the first metric is the maximum number of required flow entries (FE) on all the switches for these new bursty flows. The other two metrics mainly measure the different route performance, including deployment delay (DD) and network throughput (NT). When a larger number of bursty flows arrive in a network, since the controller needs time to send flow-mod messages and the switch needs time to install flow entries, the deployment delay represents the maximum time interval between the flow arrival time and departure time in the switch. The network throughput (NT) is defined as the transferred traffic amount of all these new bursty flows.

We run three groups of experiments on two different topologies to test the effectiveness of the proposed algorithm. As running examples, we select two practical topologies, one for datacenter networks and one for campus networks. The first topology is the fat-tree topology [22], which has been widely used in many datacenter networks. This topology is denoted by (a), contains 4 core switches, 8 aggregation switches, 8 edge switches and 16 servers. The second topology is for campus networks, denoted by (b), contains 20 switches, 40 servers and 74 links from [23]. For both topologies, the number of new bursty flows is changing from $4K$ to $16K$. The authors of [2] have shown that less than 20% of the top-ranked flows may be responsible for more than 80% of the total traffic. Thus, we allocate the size for each flow according to this 2-8 distribution. Since todays commodity switches typically contain less than $5K$ flow entries [2], we assume the already used number of flow entries on each switch is $4K$ and the available size $n(s) - n_u(s)$ for new bursty flows is $1K$ for simplicity. Note that, the set of these values has no impact on the comparing of these algorithms.

B. Flow Entries Requirement Comparison

This simulation mainly observes the number of required flow entries for these new bursty flows by different algorithms through changing the number of bursty flows on two topologies. Since MCF and RRTM should satisfy the flow table constraint on each switch, the number of required flow entries is $1K$ at most with the increasing number of flows. Meanwhile, Figs. 1 and 2 show that MCF-UN requires a larger number of flow entries with the increasing number of flows, it is because MCF-UN does not consider the flow table constraint. For example, it needs about $4K$ flow entries by MCF-UN when the network contains $14K$ flows on topology (a). Meanwhile, only about $1K$ flow entries are required by RRTM. In other words, our RRTM algorithm can reduce the required flow entries by about 75% compared with the MCF-UN algorithm on both topologies (a) and (b). The reason is that our proposed algorithm considers the flow table constraint.

C. Deployment Delay Comparison

This simulation shows how the number of flows affects the routing deployment delay on two topologies. The results of the three routing protocols are shown in Figs. 3 and 4. We can discover that the RRTM algorithm keeps the routing deployment delay about 4s, while the routing deployment delay by the MCF-UN algorithm and MCF algorithm increases with the increasing number of bursty flows significantly. For example, when there are $12K$ bursty flows in topology (b), the MCF-UN algorithm and the MCF algorithm need about 11s and 8s for the routing deployment, respectively. In other words, RRTM algorithm can reduce the routing deployment delay by about 64% and 50% compared with MCF-UN and MCF, respectively. This is because our proposed algorithm considers the controller processing capacity and forwards some flows by OSPF path, which greatly relieved the load of the controller.

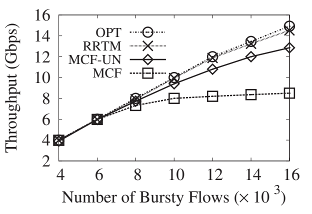

Fig. 5: Network Throughput vs. Number of Bursty Flows for Topology (a)

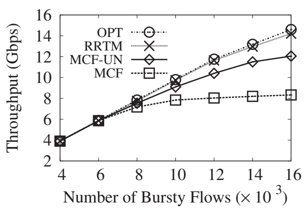

Fig. 6: Network Throughput vs. Number of Bursty Flows for Topology (b)

D. Network Throughput Comparison

Figs. 5 and 6 observe the network throughput on topologies (a) and (b), by changing the number of new bursty flows on the network. Both two figures show that the network throughput increases with the increasing number of flows. We can make some conclusions from these figures. First, both two figures indicate that the network throughput using MCF algorithm increases slowly with the increasing number of flows, that is because the limited flow table size and high-load for controller cause many flows have to be dropped. Second, we can see that the performance of OPT and RRTM is very close, which indicates our algorithm is very efficient to solve Eq. (1). Third, RRTM algorithm can achieve a better performance than MCF-UN, that is because our proposed algorithm considered the load on the controller and many flows are forwarded by OSPF path, which can decrease the load of the controller. For example, when there are $14K$ flows on the topology(b), the RRTM algorithm can increase network throughput by about 62% compared with MCF while consuming a similar number of flow entries. Besides, our RRTM algorithm can increase network throughput about 16% compared with MCF-UN while reducing the required number of flow entries by about 75%.

VI. Conclusion

In this paper, we studied the throughput maximization with limited resources (combining flow table size and controller processing capacity constraints) in SDNs. Besides, We proposed a rounding-based routing algorithm and analyzed the approximate performance. The results of simulations showed that the proposed algorithm can get better throughput performance compared with other works under the resources constraints. In the future, we will study the communication mechanism between the controller and switches, which may help to reduce the communication traffic and improve network scalability.

VII. acknowledgment

This paper is supported by the NSFC under Grant No. 61472383, 61472385 and U1301256, and NSF of Jiangsu in China under No. BK20161257.

References

[1] M. Team, "Mininet," *URL: http://mininet.org*.
[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.
[3] K. Kannan and S. Banerjee, "Compact tcam: Flow entry compaction in tcam for power aware sdn," in *International Conference on Distributed Computing and Networking*. Springer, 2013, pp. 439–444.
[4] O. F. S. Specification, "Version 1.3. 0, june 25, 2012."
[5] C. Metter, S. Gebert, S. Lange, T. Zinner, P. Tran-Gia, and M. Jarschel, "Investigating the impact of network topology on the processing times of sdn controllers," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 1214–1219.
[6] K. Kogan, S. Nikolenko, W. Culhane, P. Eugster, and E. Ruan, "Towards efficient implementation of packet classifiers in sdn/openflow," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 153–154.
[7] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.
[8] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.
[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on sdn network utilization," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1734–1742.
[10] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 934–942.
[11] Z. Hu and J. Luo, "Cracking network monitoring in dcns with sdn," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 199–207.
[12] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using openflow," *ACM Sigcomm Demo*, vol. 4, no. 5, p. 6, 2009.
[13] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 7–12, 2013.
[14] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 18–25.
[15] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *Proc. of INFOCOM*, 2016.
[16] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," *IEEE transactions on computers*, vol. 47, no. 2, p. 263, 1998.
[17] C. Wei, R. Buffone, and R. Stata, "System and method for website performance optimization and internet traffic processing," Feb. 7 2012, uS Patent 8,112,471.
[18] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Foundations of Computer Science, 1975., 16th Annual Symposium on*. IEEE, 1975, pp. 184–193.
[19] I. I. CPLEX, "V12. 1: Users manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.
[20] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
[21] "Linux foundation collaborative project," http://opendaylight.org/.
[22] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.
[23] "The network topology from the monash university," http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies.