Computer Networks 125 (2017) 26–40

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

TCAM space-efficient routing in a software defined network

Sai Qian Zhang[a,1,*], Qi Zhang[b,2], Ali Tizghadam[a], Byungchul Park[a], Hadi Bannazadeh[a], Raouf Boutaba[c], Alberto Leon-Garcia[a]

[a] Department of Electrical and Computer Engineering, University of Toronto, Canada
[b] Amazon Inc., United States
[c] David R. Cheriton School of Computer Science, University of Waterloo, Canada

ARTICLE INFO

Article history:
Received 15 October 2016
Revised 4 March 2017
Accepted 20 June 2017
Available online 5 July 2017

Keywords:
Software defined networking
Ternary Content-Addressable Memory
Traffic engineering

ABSTRACT

Software Defined Networking (SDN) enables centralized control over distributed network resources. In SDN, a central controller can achieve fine-grained control over individual flows by installing appropriate forwarding rules in the network. This allows the network to realize a wide variety of functionalities and objectives. However, despite its flexibility and versatility, this architecture comes at the expense of (1) laying a huge burden on the limited Ternary Content Addressable Memory (TCAM) space, and (2) limited scalability due to the large number of forwarding rules that the controller must install in the network. To address these limitations, we introduce a switch memory space-efficient routing scheme that reduces the number of entries in the switches, and at the same time guarantees the load balancing on link resources utilization. We consider the static and dynamic versions of the problem, analyzing their complexities and propose respective solution algorithms. Moreover, we also consider the case of fine-grained control for the flows, and develop a 2-approximation algorithm to achieve load balancing on the TCAM space usage. Experiments show our algorithms can reduce TCAM usage and network control traffic by 20% – 80% in comparison with the benchmark algorithms on different network topologies.

1. Introduction

Software Defined Networking (SDN) is an architecture that enables logically centralized control over distributed network resources. In SDN, a centralized controller makes forwarding decisions on behalf of the network forwarding elements (e.g. switches and routers) using a set of policies. Based on given high level design requirements, the source and the destination node of each flow is dictated by the *Endpoint Policy* and the flow path is decided by the *Routing Policy* [1]. For example, the shortest-path routing policy asks the network to forward packets along the shortest path between two nodes. Other routing policies that improve resource utilization, quality of service and energy usage have also been proposed in the literature [2–4]. These features make SDN an attractive approach for realizing a wide variety of networking features and functionalities.

Implementing routing policies in SDN may require fine-grained control over flows, which can place a huge burden on switch memory space. Of particular interest is the Ternary Content Addressable Memory (TCAM), a special type of high speed memory that can search the entire memory space within a single clock cycle. However, TCAM has a well known problem on limited capacity and large power consumption [5]. The largest average memory space on TCAM chip is far less than that of Binary Content Addressable Memory (CAM). For example, HP ProCurve 5406zl TCAM switch hardware can support 1500 OpenFlow rules, while each host requires dozens of OpenFlow rules on average, which means 5406zl can support only 150 users [6]. Moreover, TCAM is also energy-hungry, it consumes 30 times as much energy as SRAM with the equal number of entries [7]. As shown in Fig. 1, the energy consumption of the TCAM can contribute to up to 25% of total power required for a high-end switch ASIC [8]. Given that the amount of power consumption is proportional to the number of entries used in TCAM, a wealth of research literature is focused on reducing TCAM usage [1,5,9].

Scalability is another issue resulting from fine-grained centralized control. For every subtle change on the network topology or routing policy, the controller must deliver a control message to each network element that implements the policy. As the average flow size in both wide-area and data center networks is small (around 20 packets per flow [7]) and the inter-arrival rate

* Corresponding author.
E-mail address: sai.zhang@mail.utoronto.ca (S.Q. Zhang).
[1] The author is now at Harvard University.
[2] The work is conducted during the author's post-doctoral fellowship at University of Toronto.

http://dx.doi.org/10.1016/j.comnet.2017.06.020
1389-1286/

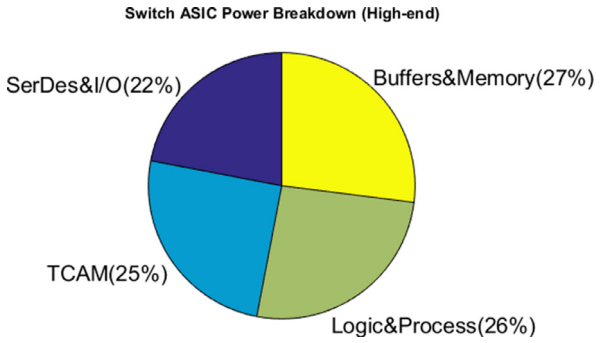


Fig. 1. Power breakdown of high-end switches ASIC.

of the flows in the high-performance network is extremely high (less than 30 ms [7]), a huge workload is imposed on the controller as the network size grows. Since each switch typically has limited bandwidth on the path to the controller, and incurs moderate rule insertion time, the high workload received by the controller often causes large rule installation overhead and leads to low flow set up rate. In modern data center networks, 1ms additional latency for the delay-sensitive flow can be intolerable [10]. Therefore, the limited flow set up rate can dramatically hurt the overall performance and the quality of service. It is important to reduce the interaction between control plane and data plane in order to achieve better network scalability and performance.

To address the issues of switch memory space limitation and scalability, recent work has proposed to control flows collectively at an aggregated level. This allows the use of prefix aggregation and wild card rules to minimize the number of stored entries [1,9]. These works have focused on compressing the entries of each individual switch, while preserving the routing policy (i.e. without changing the forwarding paths) [6]. However, we find that in large networks, multiple candidate paths are usually available for routing each individual flow while still satisfying performance and business constraints. Therefore, if we can additionally control the flow forwarding paths, we can achieve substantial gains in terms of TCAM space savings and controller scalability. To this end, we propose a new routing scheme that minimizes TCAM space consumption in SDN networks without causing network congestion. The proposed routing scheme takes advantages of the large number of available forwarding paths and routes traffic in a way that improves network scalability and reliability. The main objectives of this new routing scheme are:

- Minimize the switch memory space utilization for a given the end point connection request.
- Reduce control traffic by decreasing the interaction between the controller and network.

To this end, we show that by appropriately using the subnet masks on the address field, we can achieve significant saves in the TCAM space while guaranteeing the load balancing on link resources.

In this paper, we first introduce the TCAM space minimization problem and analyze its complexity. We then propose heuristic algorithms for both static and dynamic versions of the problem. Through experiments, we show that our algorithms reduce the TCAM space usage and network control traffic by 20% – 80% in comparison with the benchmark algorithms.

The rest of the paper is organized as follows. Section 2 reviews the related works. Section 3 motivates our problem through a descriptive example. Section 4 presents the problem statement, and Section 5 formulates a corresponding a traffic engineering problem, and show the problem is NP-hard and inapproximable in general. We then propose a heuristic to solve the traffic engineering problem. Section 6 introduces partitioning of all the demand pairs into groups to achieve minimum TCAM space utilization. Section 7 provides an online algorithm to deal with the dynamic entry and departure of the demand pairs. Section 8 presents and solves the problem of the efficiently placement of the rules to realize fine-grained control. Section 9 presents performance results from simulations of the proposed algorithms. Section 10 evaluates the routing scheme on real testbed and discusses potential implementation issues. Section 11 presents conclusions.

2. Background and related work

OpenFlow is a popular and efficient means for realizing a centralized control framework [11]. It allows the controller to choose the paths of packets across a set of switches. An OpenFlow table entry in a switch can be represented by a triplet (*M, P, A*) [11], where *M* is the matching field which is used to match the packet, *P* is the matching precedence of the entry and *A* is the action field which contains operations on the matched packet. The matching field usually includes source IP address, source MAC address, destination IP address, destination MAC address, input port number. The action field includes common operations such as forwarding the packet to a specific output port, modifying the packet header, etc. Upon receiving a packet, the switch searches for the rule with the highest priority that matches the packet, then executes the corresponding actions defined by that rule. OpenFlow also supports wildcard over the input port region and subnet mask in the IP and MAC address to represent a group of source and destination IP/MAC addresses [11], for instance, 01** in the address field stands for 0100, 0101, 0110 and 0111.

To deal with the TCAM space issue, previous works have focused on compressing the entries of a single switch, guaranteeing that the overall forwarding logic of that switch keeps the same, while preserving the routing policy [5]. One Big Switch [1] and Palette [9] decompose network access policies into small parts and then distribute these to use TCAM space. Moshref et al. [12] designs routing algorithms to distribute access policies across intermediate switches with minimum switch memory consumption in a datacenter network. Rami et al. [25] study the effect of flow table size on the maximum number of flows supported. CacheFlow [26] develops an algorithm for placing rules in a TCAM with a limited space.

Scalability is a key issue in SDN. DevoFlow [6] presents a scalable SDN framework by using wildcard entries to decrease the control plane visibility on the microflows. However, it does not offer sufficient quantitative analysis about how to use the wildcard to achieve optimal performance. DIFANE and Kandoo [13,14] propose efficient and scalable SDN frameworks which split the workload of the central controller to distributed authorized components. However, the problem of global visibility has not been tackled. The authors of [15,16] solve the scalability issue by using multiple independent controllers to consistently manage the whole network, while minimizing the amount of communication between them, but they do not address how these controllers are coordinated and communicate with each other, and they ignore the overhead brought by distributing the control protocol.

3. A motivating example

We provide a motivating example to demonstrate the benefit of the proposed switch memory space-efficient routing scheme. A network topology and port numbers between nodes are shown in Fig. 2(a) and the end point policy is shown in Fig. 3(a). Two source hosts with the IP address 000 and 001 send traffic to two destination hosts 100 and 101 respectively. We call a pair of source and destination address a demand pair, and so there are four demand

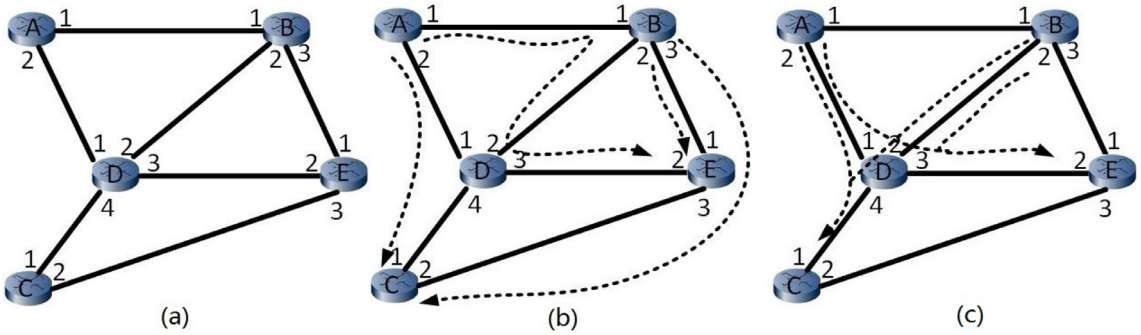


Fig. 2. Motivation example.

(a)

Srcip	Dstip	Ingress	Egress
000	100	A	C
001	101	B	E
001	100	B	C
000	101	A	E

(b)

Node	Src	Dst	Inport	Act.
A	000	101	-	1
	000	100	-	2
B	000	101	1	2
	001	101	-	3
	001	100	-	3
C	000	100	1	-
	001	100	2	-
D	000	100	1	4
	000	101	2	3
E	001	101	1	-
	001	100	1	3

(c)

Node	Src	Dst	Inport	Act.
A	0**	1**	-	2
B	0**	1**	-	2
C	0**	1**	1	-
D	000	100	*	4
	001	100	*	4
	000	101	*	3
	001	101	*	3
E	0**	1**	2	-

Fig. 3. Example of flow tables.

pairs in this example. The bandwidth consumption of each demand pair equals 1 and the capacity of each link is 10. Traditional traffic engineering (e.g. ECMP) spreads the flows evenly in the network to balance network link utilization, which gives one feasible solution in Fig. 2(b).

The OpenFlow table of each switch is shown in the Fig. 3(b). A total of 11 entries are installed. To set up these new flows, 11 additional control packets are sent from the controller, since at least one initial packet in each new flow is processed by the controller. In total $11 + 2 \times 4 = 19$ packets are transmitted between the controller and the switches. In contrast, the proposed routing scheme produces the solution in Fig. 2(c) and the forwarding tables shown in Fig. 3(c). Instead of routing the traffic of each demand pair respectively, this routing scheme aggregates the flows and uses subnet masks to reduce the number of entries in each table. The maximum bandwidth consumption of the solution given by the new scheme is also 2, and 8 additional entries are installed on nodes $A - E$, which requires 8 control packets sent from controller. A total $8 + 2 \times 4 = 16$ packets are transmitted between controller and switches. This reduces TCAM space and control traffic by 27.2% and 15.8% respectively. From the above example, we draw the following conclusions:

1. If fine-grained control is not required on specific flows, TCAM space consumption and control traffic can be reduced by using subnet masks on the source and destination addresses to aggregate flow entries.
2. As the network size increases, the number of control packets to set up a flow is approximately equal to the number of entries installed in the TCAM (ignoring the initial packet of the flow that is sent to the controller). So minimizing TCAM usage can also indirectly save control traffic indirectly.
3. In addition to finding a path which minimizes TCAM consumption, the constraint on link capacity must also be met. For example, the two solutions above have the same maximum link utilization.

4. Problem overview

The design objective of the proposed switch memory space-efficient routing scheme is to minimize the total number of OpenFlow entries installed in all the switches, which is equivalent to minimizing the average number of entries installed in each switch. To keep the problem generic, we assign a weight to each switch in the network, where this weight is the cost of installing an additional rule in the switch. For the choice of $w(v)$, in the simplest case, we can set $w(v) = 1$ to achieve the goal of minimizing total number of forwarding entries in the switches. However, adjusting the value of $w(v)$ allows us to model other objectives. For instance, since power consumption of a switch is linearly proportional to the TCAM space usage [5], by setting $w(v)$ to the average power consumption per rule for switch v, we can model the problem of minimizing total energy consumption in the network. The objective then would be to minimize the total weighted cost, given a set of demand pairs and constraints on link resource utilizations. We call this the *TCAM Space Minimization Problem (TSMP)*.

TSMP is a rather complex problem to analyze and solve directly. To simplify our analysis, we divide TSMP into two sub-problems: *Efficient Partitioning Problem (EPP)* and *Efficient Routing Problem (ERP)*. The EPP focuses on partitioning all the demand pairs into groups. We call these groups the *routing groups*. The source addresses and destination addresses in the same routing group have common prefixes. For example, the four demand pairs [000, 100], [000, 101], [001, 100], [001, 101] in the Fig. 3(a) form a routing group with prefix 0^{**} and 1^{**}, where we use $[s_k, d_k]$ to represent the demand pair.

We can use the addresses with subnet mask $s_u = 0^{**}$ and $d_u = 1^{**}$ to represent all the source addresses and destination addresses in the routing group u. When partitioning is complete, for each routing group there will be a corresponding ERP, and we route all demand pairs in that routing group to minimize total TCAM space usage.

Table 1
Definitions of parameters.

Name	Description	Name	Description
G	A network topology $G = (V, E)$	V	Set of nodes in G
E	Set of links in G	S	Set of addresses of source hosts
D	Set of addresses of destination hosts	U	denote the set of routing groups
K	Set of demand pairs	K_u	Set of demand pairs in the routing group $u \in U$
m	Number of bits in the source address and destination address	$size(u)$	The number of demand pairs in routing group u
s_k	the source address of demand pair k	d_k	the destination address of demand pair k
s_u	The source address of routing group u with the subnet mask	d_u	The destination addr. of routing group u with the subnet mask
$a(v)$	Number of OpenFlow rules installed on switch v	$w(v)$	Cost of inserting a single OpenFlow rule in switch v
r_v	The TCAM space capacity of switch v	$\pi(v)$	Set of port numbers associated with switch v
$p(v)$	Set of port number pairs of switch v	β	Threshold of link utilization rate
B_k	The bandwidth consumption of $k \in K$	C_e	C_e the capacity of each link $e \in E$
x_{ijk}	A binary variable, $x_{ijk} = 1$ if an 4-tuple (s_u, i, d_u, j) is installed to direct traffic of demand pair k from port i to port j, $x_{ijk} = 0$ otherwise	y_{ij}	A binary variable, $y_{ij} = 1$ if a 4-tuple (s_u, i, d_u, j) is installed to direct the flow of $s_u \in S$ from port i to port j and $y_{ij} = 0$ otherwise
l_{ek}	A binary variable, $l_{ek} = 1$ denotes edge $e \in E$ is used to direct the flow of demand pair k	K_{fine}	The set of demand pairs required to gather statistics
L	Maximum number of demand pairs in each routing group	z_{vk}	A binary variable, $z_{vk} = 1$ if the rule is installed on switch v to gather the statistics for specific flow of $k \in K_{fine}$
λ	The TCAM space utilization rate	q_v	The initial number of rules installed on switch v before the rules for collecting specific flow statistics is added
$H(k)$	The path of the demand pair $k \in K_{fine}$		

Based on the description of ERP and EPP above, we can formulate TSMP: let U denote the set of routing group, K denote the set of routing pairs and $K_u (u \in U)$ denote the set of demand pairs in u (Table 1 provides a quick glossary of definitions). Furthermore, define $TCAMcost(K_u)$ to be the minimum cost returned by ERP to route the demand pairs in K_u. TSMP can be formulated as:

$$\underset{u}{\text{minimize}} \sum_{u \in U} TCAMcost(K_u) \tag{1}$$

$$s.t. \bigcap_{u \in U} K_u = K \tag{2}$$

The main challenge here is that the EPP and ERP are not independent. The routing groups given by the solution of the EPP will determine the input of ERP, which determines the total amount of switch memory space consumed.

In the next two sections we first discuss our algorithm for ERP, and then the solution for EPP, which relies on the solution algorithm for ERP to make partitioning decisions.

5. Efficient routing problem

The goal of ERP is to connect each demand pair for a given routing group while consuming minimum weighted sum of switch memory space and satisfying the load balancing on links. Formally, we model the network as a graph $G = (V, E)$, where each node $v \in V$ represents an OpenFlow switch and each switch v is assigned a cost $w(v)$ per rule inserted. Without loss of generality, we assume each flow entry in the flow table can be represented by a 4-tuple (s, i, d, j), where s, i, d constitute the matching field: s, d represent the source and destination address information, such as source/destination IP/MAC address, i is the input port number of the switch where the packet comes in. j is the output port number of the switch that the packet is directed to, and which constitutes the action field of the OpenFlow entry. We neglect rule priority for now and consider it later.

Let s_k and d_k denote the source and destination addresses of demand pair k. We use a 4-tuple (s_u, i, d_u, j) to represent the OpenFlow rule installed for the routing group $u \in U$, where s_u and d_u are the source and destination addresses with the subnet masks respectively.

Let $\pi(v)$ be the set of port numbers of switch v. We make the port number equal to the label of the links that the port connects to (Fig. 4). Then we denote $p(v) = \{(x, y) : x \in \pi(v), y \in \pi(v)\}$ as

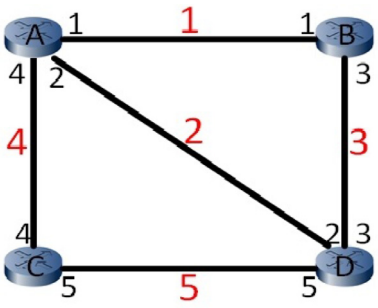


Fig. 4. Labelling port example.

the set of port pairs of switch v. For example $\pi(A)$ in Fig. 4 is {1, 2, 4} and $p(A) = \{(1, 2), (2, 1), (1, 4), (4, 1), (2, 4), (4, 2), (1, 1), (2, 2), (4, 4)\}$. Let $y_{ij} \in \{0, 1\}$ represent whether a 4-tuple is installed to direct the flow of routing group u from input port i to output port j. Let $x_{ijk} \in \{0, 1\}$ denote whether a 4-tuple entry is installed to direct traffic of a demand pair $k \in K_u$ from input port i to output port j. Let $a(v)$ denote the total number of rules installed on switch v. Our goal is to minimize the total weighted sum of rules installed in the switches:

$$\underset{x_{ijk}, y_{ij} \in \{0,1\}}{\text{minimize}} \sum_{v \in V} w(v) a(v) \tag{3}$$

where $a(v)$ represents the number of 4-tuples (s_u, i, d_u, j) installed in v. To compute $a(v)$, note that for the same switch v and same routing groups, three conditions may occur:

- No 4-tuple (s_u, i, d_u, j) needs to be installed on v. That is, $\sum_{j \in \pi(v)} \mu(\sum_{i \in \pi(v)} y_{ij}) = 0$ and therefore $a(v) = 0$, where μ is the step function, $\mu(x) = 0$ if $x \leq 0$ and $\mu(x) = 1$ if $x > 0$.
- All the flows installed on v are forwarded to one output port, i.e., $\sum_{j \in \pi(v)} \mu(\sum_{i \in \pi(v)} y_{ij}) = 1$. One entry $(s_u, *, d_u, j)$ is enough to direct the flows of K_u, with s_u and d_u in the address field and wildcard in the input port field, so $a(v) = 1$.
- All the flows installed on v are forwarded to more than one output port. That is, $\Sigma_{j \in \pi(v)} \mu(\Sigma_{i \in \pi(v)} y_{ij}) > 1$, therefore, the source and destination fields must be fully specified to differentiate each flow and so that the flows can be directed to corresponding output ports. Hence the total number of entries installed is $\sum_{i \in \pi(v)} \sum_{j \in \pi(v)} \sum_{k \in K_u} x_{ijk}$, which is the number of demand pairs whose flows traverse through v.

Switch	Src	Dst	Inport	Action
s1	00	10	1	output:4
	00	11	1	output:3
	01	10	2	output:5
	01	11	2	output:6

Fig. 5. Flow Table of s1.

The conditions can be illustrated by the following example: Assume a set of rules {(00, 1, 10, 4), (01, 2, 10, 5), (00, 1, 11, 3), (01, 2, 11, 6)} is installed on $s1$. The flow table of $s1$ is shown in Fig. 5. As the table shows, the source and destination address must be fully specified so that each flow can be identified by the intermediate switch to direct to its corresponding output port.

By combining the 3 cases, $a(v)$ can be defined as follows:

$$a(v) = \begin{cases} 0 & \text{if } \sum_{j\in\pi(v)} \mu(\sum_{i\in\pi(v)} y_{ij}) = 0 \\ 1 & \text{if } \sum_{j\in\pi(v)} \mu(\sum_{i\in\pi(v)} y_{ij}) = 1 \\ \sum_{i\in\pi(v)} \sum_{j\in\pi(v)} \sum_{k\in K_u} x_{ijk} & \text{if } \sum_{j\in\pi(v)} \mu(\sum_{i\in\pi(v)} y_{ij}) > 1 \end{cases}$$

We can ensure that the number of rules installed in each switch does not exceed its TCAM space capacity: let r_v be the capacity of switch v, we then require:

$$a(v) \le r_v \quad \forall v \in V \tag{4}$$

Next, we relate x_{ijk} to y_{ij}. Eq. (5) ensures that 4-tuple rule (s_u, i, d_u, j) is installed if any flow of demand pair k is sent from input port i to output port j, *and so*:

$$\sum_{k:k\in K_u} x_{ijk} \le y_{ij} \quad \forall (i,j) \in p(v), v \in V \tag{5}$$

Next we build the path between each source host to the destination host. Let $l_{ek} \in \{0, 1\}$ denote whether edge $e \in E$ is used to direct the flow of demand pair k. Define $Q_k = \{Q_k \subseteq V : s_k \in Q_k, d_k \notin Q_k\}$ $(\forall k \in K)$ and define $\pi(Q_k)$ the set of edges in the *cut* defined by Q_k, that is, the set of edges in G which have ingress node in the set Q_k. Then we have:

$$\sum_{e:e\in\pi(Q_k)} l_{ek} \ge 1 \quad \forall k \in K_u \tag{6}$$

By *max-flow/min-cut* theorem, Eq. (6) ensures there exists at least one path between s_k and d_k [17]. Next the following equations make sure OpenFlow entries are installed to direct the flow to each used link:

$$l_{ek} \le \sum_{v\in V} \sum_{i:(i,e)\in p(v)} x_{iek} \le 1 \quad \forall e \in E, k \in K_u \tag{7}$$

$$l_{ek} \le \sum_{v\in V} \sum_{j:(e,j)\in p(v)} x_{ejk} \le 1 \quad \forall e \in E, k \in K_u \tag{8}$$

$$\sum_{(i,j)\in p(v)} x_{ijk} \le 1 \quad \forall k \in K_u, v \in V \tag{9}$$

Eqs. (7)–(9) ensure that if link e is used to direct the flow for k, then there exists exactly one flow entry in the ingress switch of e to direct the flow of k to e and there exists one flow entry in the egress switch of e to accept the flow of k from link e. Finally, we must consider the constraint on maximum bandwidth utilization rate on all links. Define B_k as the bandwidth consumption for the demand pair k, C_e the capacity of each link $e \in E$, and let β be the limit on link utilization rate. we have:

$$\sum_{k:k\in K} B_k l_{ek} \le \beta C_e \quad e \in E \tag{10}$$

The goal of ERP is to minimize objective function (3), subject to Eqs. (4)–(10).

Next we consider the complexity of ERP. Theorem 1 shows the NP-completeness and inapproximability of the *ERP* which implies the NP-completeness and inapproximability of *TSMP*. Theorem 2 shows that even without the load balancing guarantee (10), or some other performance guarantee than (10), the *ERP* is still NP-hard and $(1-\epsilon)\ln|V|$ inapproximable for any $\epsilon > 0$.

Theorem 1. *ERP is NP-complete and inapproximable.*

Proof. The proof is based on reduction from the *3-partition problem*[3]. Consider a part of a hierarchical tree topology in a datacenter in Fig. 6(a). Four source hosts inject packets to *A, B, C, D*, and the bandwidth consumption B_k of the traffic injected on *A, B, C, D* are b_1, b_2, b_3, b_4 respectively. The maximum usage on bandwidth βC_e of link (*E, H*), (*G, H*) and (*F, H*) equal $\frac{1}{3}(b_1+b_2+b_3+b_4)$. To satisfy (10), the flows from the four source nodes must be partitioned into three subsets with the same total amount of bandwidth $\frac{1}{3}(b_1+b_2+b_3+b_4)$. Therefore by knowing whether the problem is feasible or not, we know whether the set of numbers $\{b_1, b_2, b_3, b_4\}$ can be partitioned into three subsets with the equal sum of elements. Since the decision version of 3-partition problem is NP-complete, then any polynomial-time approximation algorithm for this problem would solve the 3-partition problem in polynomial time, which is not possible unless $P = NP$. □

Following the same arguments, we can also show that *TSMP* is also NP-complete and inapproximable.

Theorem 2. *Even without the link capacity constraints (i.e., Eq. (10)), ERP defined by* (3) – (9) *is NP-hard, and there is no* $(1-\epsilon)\ln|V|$*-approximation algorithm for any* $\epsilon > 0$*, where* $|V|$ *is the number of nodes in G.*

Proof. The proof is based on a reduction from the *set cover problem*[4]. Consider a multi-root hierarchical tree topology in Fig. 6(b), where each node on layer 3 does not fully connect to every node on layer 2 due to link failure. 4 source hosts form a routing group, and each connects with the switches *A, B, C, D* and sends traffic to the core switch *H*. Assume r_v is large and the weight of all the switches on layer 3 and layer 1 is small, then the objective functions (3) is equivalent to minimizing the number of entries inserted on layer 2 switches.

Since each additional switch used in layer 2 to direct the flow from $A-D$ corresponds to an additional flow entry inserted on that switch, then minimizing number of entries on layer 2 switches is equivalent to minimizing the number of switches used on layer 2. Define the universal set $U = \{A, B, C, D\}$ to consist of all the layer 3 switches, and assign a subset of U to each switch on layer 2. The subset for each switch on layer 2 consists of the switches on layer 3 that the switch connects to. For example, the subset for $E = \{A, C\}$ and the subset for $F = \{B, D\}$. In order to make sure there is a path from $A-D$ to destination H, we need to ensure each switch on layer 3 connects to at least one switch on layer 2. Therefore, minimizing the number of additional flows inserted on layer 2 switches is equivalent to minimizing the number of layer 2 switches used to direct the flow, which is equivalent to minimizing the number of subsets used to cover the universal set U, which in turn is the definition of set cover problem. Since the set cover problem is NP-hard and cannot be approximated with in a factor

[3] The partition problem is the task of deciding whether a set of positive integers can be partitioned into three subsets *X, Y* and *Z* such that the sums of the numbers in *X, Y, Z* are equal.

[4] Given a set of elements {1,2,...,m}, and a set *A* of *n* sets whose union equals the element set, the set cover problem is to find the smallest subset of *A* whose union contains every single element.

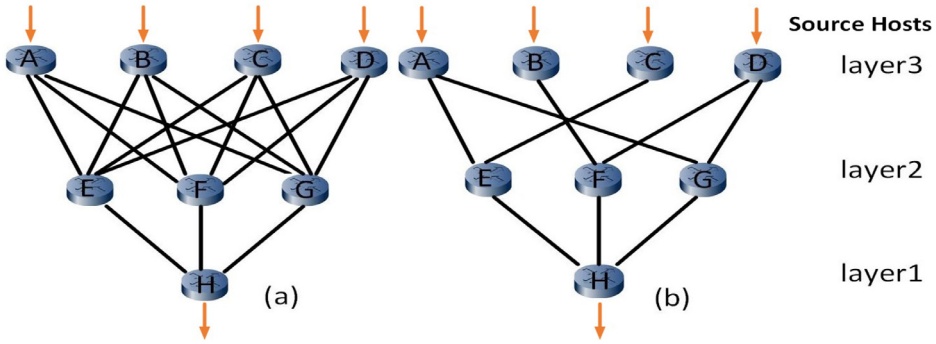


Fig. 6. Proof of inapproximability.

of $(1-\epsilon)\ln n$ for any $\epsilon > 0$ (where n is the size of the set), the ERP is also NP-hard and $(1-\epsilon)\ln|V|$ inapproximable for any $\epsilon > 0$. □

Since ERP is both NP-complete and inapproximable, we propose a simple and efficient heuristic to solve ERP. Without loss of generality, given an undirected topology $G = (V, E)$ the graph can be made directed by replacing each undirected link e by two directed links e' with opposite directions, where we mark both directed links by e' evolved from e. We define a new directed graph $G' = (V', E')$, and $in(e')(e' \in E')$ as the *ingress switch* (head) of e' and $out(e')(e' \in E')$ as the *egress switch* (tail) of e'. An directed link e' is a link from its *egress switch* (tail) to its *ingress switch* (head). Define $C_{e'}(e' \in E')$ as the capacity of the link e', which equals that of C_e, where e is the undirected link from which e' is created. We relate the cost of inserting rules on switches to the weight of the directed links of the switches. First, we provide the following definition:

Definition 1. Link e' is *ready* for routing group u if: 1. $out(e')$ contains a 4-tuple (s_u, i, d_u, e'), $i \in \pi(out(e'))$ or $(s_u, *, d_u, e')$. 2. $in(e')$ contains a 4-tuple (s_u, e', d_u, j), $j \in \pi(in(e'))$ or $(s_u, *, d_u, j)$.

In other words, a link is *ready* for u if there already exists an entry on its ingress switch and egress switch to forward the flow onto this link. Next we calculate the cost of activating the links e' on switch $out(e')$. Let $t(v)(v \in V)$ be the number of demand pairs of u that v carries after the e' is activated. Define θ_v^u the number of egress links of v used to direct the traffic of demand pairs of u before e' is added. Then the cost of activating this link e', $cost(e')$ is shown below:

$$cost(e') = \begin{cases} w(out(e')) & \text{if } \theta^u_{out(e')} = 0 \text{ or } \theta^u_{out(e')} > 1 \\ (t(out(e')) - 1)w(out(e')) & \text{if } \theta^u_{out(e')} = 1 \quad (\star) \end{cases}$$

For each newly activated link e', the corresponding OpenFlow rule has to be installed to the $out(e')$ to direct the traffic. If initially no other link of $out(e')$ is used, one OpenFlow entry $(s_u, *, d_u, n(e'))$ will be installed on $out(e')$, so $cost(e') = w(out(e'))$. However, if previously one egress link has been activated on switch $out(e')$, then initially all the flows are forwarded to single output port. To activate a new link with a new output port, we now require the all the flows carried by the switch to be fully specified so that they can be directed to the corresponding output ports. Hence $cost(e') = (t(v) - 1)w(out(e'))$. Finally, if previously more than one link has been activated on switch $out(e')$, for each new activated egress link, a new corresponding entry $(s_k, i, d_k, n(e'))(k \in K_u)$ is installed to direct the flow.

An example is given in Fig. 7(a): Assume initially switch $s1$ carries two demand pairs [00, 10] and [01, 11] of u that have the same output port 4 ($\theta_v^u = 1$), therefore one entry is installed to route the flows as shown in Fig. 7(b). Now assume one more demand pair [00, 10] is added and another egress link is used to direct this flow (output port is 5), then number of entries in the routing table increases by $t(v) - 1 = 3 - 1 = 2$. Therefore the cost to activate this new link is $2w(v)$, the new flow table is shown in Fig. 7(c).

Algorithm 1 Incremental routing algorithm (IRA).

```
1:  for each demand pair k ∈ K_u do
2:    for each link e' ∈ E' do
3:      if e' is ready for k then
4:        Set the cost of link e' to 0, cost(e') = 0
5:      if e' is not ready for k then
6:        Update the link cost cost(e') according to (⋆)
7:      if βC_e' ≤ B_k or a(out(e')) > r_out(e') then
8:        Set the cost of link e' to infinity, cost(e') = ∞
9:    Find shortest path between s_k and d_k, if there are more than
      one shortest paths, randomly select one. Install the 4-tuple
      rules along the path. Update a(v).
10:   Set βC_e' = βC_e' − B_k
```

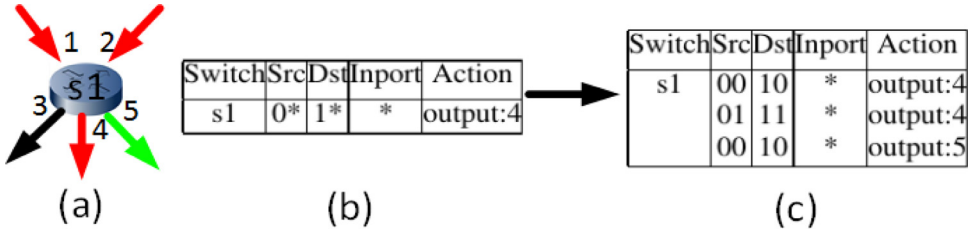
Algorithm 1 reuses the links which are *ready* by setting the weights of these links to 0. The weights of other links are updated according to $(\star)$. If the bandwidth consumption on e' exceeds the maximum limit $\beta C_{e'}$, the cost of e' is set to be infinity, $cost(e') = \infty$. Finally the solution path can be calculated by finding the shortest path between the source and the destination hosts.

We now analyze the complexity of *IRA*. The for loop between line 3 to 8 in *IRA* determines the cost for each edge $e \in E$. In line 9, the shortest path is calculated between each s_k to d_k. Therefore, the overall complexity is $\mathcal{O}(|K_u|(|V| + |E|\log|E|))$, where $|K_u|$ is the size of K_u, $|V|$ is number of nodes in the network and $|E|$ is number of edges in the network.

6. Efficient partitioning problem

After solving ERP for each routing group, we are still left with the problem of partitioning K demand pairs into routing groups. In this case all demand pairs can be visualized using a $2^m \times 2^m$ square, where m is the number of bits in the source and destination address. For example, Suppose there are 6 demand pairs [10, 00], [11, 00], [00, 01], [00, 11], [01, 11], [01, 10], the corresponding square is shown in Fig. 8(a). The squares representing the 6 demand pairs are coloured in blue. One of the possible partitions is shown in Fig. 8(b), where the routing group $G1$ covers the demand pairs [01, 10], [01, 11], [00, 11], $G2$ covers [10, 00], [11, 00] and $G3$ covers [00, 01].

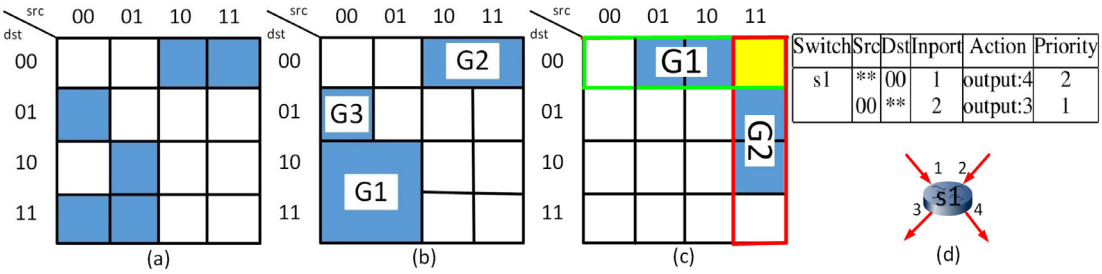
The goal of *EPP* is to find the routing groups so each group can be routed with the lowest cost as defined in Eq. (3). We represent each routing group by a pair of source-destination addresses with subnet mask. For example, $G2$ in Fig. 8(b) can be represented by [1*, 00].



Switch	Src	Dst	Inport	Action
s1	0*	1*	*	output:4

Switch	Src	Dst	Inport	Action
s1	00	10	*	output:4
	01	11	*	output:4
	00	10	*	output:5

Fig. 7. Example on cost of link.



Switch	Src	Dst	Inport	Action	Priority
s1	**	00	1	output:4	2
	00	**	2	output:3	1

Fig. 8. Examples.

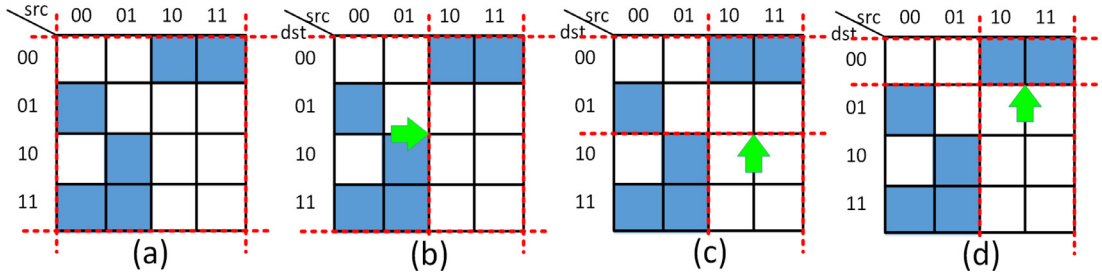


Fig. 9. Example of *DSA*.

A drawback of aggregating flow entries is that we lose visibility into the fine-grained flow characteristics, which makes elephant flow detection and rerouting harder to achieve [6]. Consequently, we impose a maximum routing group size L to limit the maximum flow aggregation level, which allows the trade-off between flow visibility and TCAM space savings.

Our solution algorithm, called *Detailed Search Algorithm (DSA)*, begins with the entire square that covers all source-destination pairs. In each iteration, it reduces the size of the routing rectangle by replacing the wildcard bit in the address with a binary digit. *The output of each iteration is the routing group with the lowest average cost per demand pair in the group*. Define the *leading bit* of an address as the leftmost wildcard bit in the address. For example, the leading bit of address 00** is the third bit. If there is no wildcard bit in the address, set the leading bit to 0. Denote $size(u)$ the number of demand pairs in routing group g. Define l_s and l_d as the leading bits of the source and destination address. The pseudo code of *Detailed Search Algorithm* is described in Algorithm 2.

The function $IRAcost(u)$ returns the minimum cost generated by *IRA* to route all the demand pairs in u. The *DSA* algorithm works by searching the routing group u' with $size(u') < L$ with the lowest average cost in a greedy fashion, and building the paths for that group with minimum cost. Subsequently, the demand pair is removed from K. The algorithm terminates when all the demand pairs in K have been routed.

Fig. 9 provides an example to illustrate *DSA*. Let L equal 3. Initially there are 6 demand pairs. The routing group is the region circled by the red dash line, which is the whole square shown in Fig. 9(a). Assume we found that the routing group with minimum average cost is [1*, **], by setting the leading bit of source address to 1, the corresponding routing group is shown in Fig. 9(b). We repeat these steps until we have found the routing group [1*, 00] shown in Fig. 9(c) and 9(d) (Note that further dividing of this routing group will increase the average routing cost per demand pair). Then the two demand pairs in the routing group [1*, 00] are routed by using *IRA*. *DSA* then removes this routing group, and we repeat the process until all the demand pairs are routed.

We now analyze the complexity of *DSA*. The inner while loop between lines $4-15$ runs at most $2m$ times, since in each iteration of the inner while loop the leading bit of source address or destination address decreases by 1, the iteration will stop when all the wildcard bits in source address and destination address are filled with binary digits. For each inner while loop, the *IRA* is called 4 times (line $5-8$). Finally, the outer wile loop (line $2-15$) runs at most $|K|$ times. Therefore the complexity of *DSA* is $\mathcal{O}(8m|K|^2(|V|+|E|\log|E|))$.

6.1. Rule priority between routing groups

It is possible that two routing groups may overlap with each other. For the example shown in Fig. 8(c), two routing groups $G1$ and $G2$ both cover the yellow square [11, 00]. Assume the switch $s1$ carries the traffic of both routing groups, the flow of [11, 00] will satisfy the predicates for both entries, which is shown in Fig. 8(d). Therefore each entry in the switch must be assigned a priority level. Upon receiving a packet, the switch finds the entries with a matching predicates and the highest priority level, and then performs its action. One simple way to assign priorities in *DSA* is based on the order the routing group is generated by *DSA*. For ex-

Algorithm 2 Detailed search algorithm (DSA).

Set the source and destination address to the address with fully wildcard bit, set $u_{prev} = u_{curr} = \emptyset$, set $l_s = l_d = m$.
while $K \neq \emptyset$ **do**
  Set $prev = \infty$ and $curr = 0$
  while $curr \leq prev$ or $size(u_{curr}) > L$ **do**
    Set $[u_{s0}, avgcost(u_{s0})] = FindCost(src, l_s, 0)$
    Set $[u_{s1}, avgcost(u_{s1})] = FindCost(src, l_s, 1)$
    Set $[u_{d0}, avgcost(u_{d0})] = FindCost(dst, l_d, 0)$
    Set $[u_{d1}, avgcost(u_{d1})] = FindCost(dst, l_d, 1)$
    Select u_{curr} equals to $u \in \{u_{s0}, u_{s1}, u_{d0}, u_{d1}\}$ with the minimum $avgcost(u)$, if more than one such u exist or all the $avgcost(u)$ equals infinity, randomly pick one.
    Set $curr = avgcost(u_{curr})$
    if ($curr > prev$ or $l_s = l_d = 0$) **then**
      Remove all the demand pairs in u_{prev} from K, building the path for each demand pair in u_{curr} by using *IRA*.
      Set the source and destination address to full wildcard bits. Set $u_{prev} = u_{curr} = \emptyset$, $l_s = l_d = m$
      break
    Set the binary digit on leading bit according to u_{curr}, update the leading bit by decreasing l_s or l_d by 1 according to u_{curr}, set $prev = curr$, $u_{prev} = u_{curr}$
Function *FindCost* (type, l, d)
if ($type == src$ and $l_s \neq 0$) **then**
  Set the binary digit on leading bit l of source address to d, while keeps destination address the same. Denote the routing group formed u.
  if ($0 < size(u) \leq L$) **then**
    Reset the binary digit on the leading bit l of the source address to wildcard bit.
    Return $[u, \frac{IRAcost(u)}{size(u)}]$
  if ($size(u) > L$) **then**
    Return $[u, \infty]$
if ($type == dst$ and $l_d \neq 0$) **then**
  Set the binary digit on the leading bit l of the destination address to d, while keeps the source address the same. Denote the routing group formed u.
  if ($0 < size(u) \leq L$) **then**
    Reset the binary digit on the leading bit l of the destination address to wildcard bit.
    Return $[u, \frac{IRAcost(u)}{size(u)}]$
  if ($size(u) > L$) **then**
    Return $[u, \infty]$
Return $[\emptyset, \infty]$
EndFunction

ample, if $G1$ is generated before $G2$, then the entry of $G1$ has a higher priority than that of $G2$ (shown in Fig. 8(d)).

7. Dynamic scheduling of demand pairs

The algorithms presented in the previous sections have been focused on the static version of the problem. While they are useful for networks that have constant network demand, in reality, the demand pairs may join/leave the network dynamically. In this section we propose the dynamic algorithms to deal with this scenario.

7.1. Dynamic demand pairs entering

We first consider the case where a new demand pair k enters the network. Let s_k and d_k denote the source and destination address of k. We first make the following definition:

Definition 2. Let f be a full address without wildcard bits, we say the address f' *covers* f if f' and f have the same bit length and all the non-wildcard bits of f' are the same as f.

For example, let $f' = 00^{**}$ and $f = 0001$, then f' covers f because all the non-wildcard bits of f' (the first two bits) are the same as f, which is 00. Next we extend the definition of *ready* for each new demand pair k:

Definition 3. In a directed graph $G' = (V', E')$, link e' is *ready* for the new demand pair k if: 1. $out(e')$ contains a 4-tuple $(s, i, d, n(e'))$, $i \in \pi(out(e'))$ or $(s, *, d, n(e'))$. 2. $in(e')$ contains a 4-tuple $(s, n(e'), d, j)$, $j \in \pi(in(e'))$ or $(s, *, d, j)$, where s *covers* s_k and d *covers* d_k.

Algorithm 3 Dynamic algorithm for new arrivals (DANA).

for each new demand pair k **do**
  for each link $e' \in E'$ **do**
    if e' is ready for k **then**
      Set the cost of link e' to 0, $cost(e') = 0$
    if e' is not ready for k **then**
      Set the link cost $cost(e') = w(out(e'))$
    if $\beta C_{e'} \leq B_k$ or $a(out(e')) > r_{out(e')}$ **then**
      Set the cost of link e' to infinity, $cost(e') = \infty$
  Find shortest path between s_k and d_k, if there are more than one shortest paths, randomly select one. Install the 4-tuple rules (s_k, i, d_k, j) along the path. Update $a(v)$.
  Set $\beta C_{e'} = \beta C_{e'} - B_k$

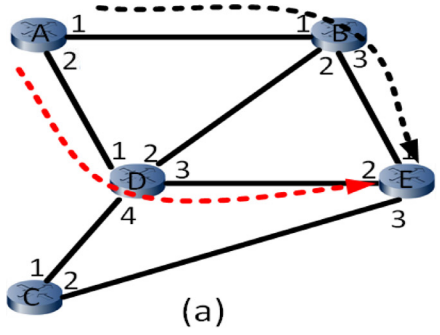
Algorithm 3 (*DANA*) builds the paths for each new demand pair. The intuition behind *DANA* is reusing existing rules in the network. For the example shown in Fig. 2(c), the routing tables are shown in Fig. 3(c). Assume that there exists a new demand pair with source address/destination address 010/101 and ingress/egress switches are A and E. Further assume that the every link has enough remaining capacity to carry the flow of this demand pair such that (10) is obeyed. Also assume that every switch has the same weight and enough TCAM space. One of the possible solutions is routing through the path A, B, E (the black route in Fig. 10(a)), and the new routing table is shown in Fig. 10(b). Three entries are added on the switch A, B and E. *DANA* will generate the red route shown in Fig. 10(a) and the routing table shown in Fig. 10(c). By comparison, only one entry is installed on switch D, and the entries in switch A, D are reused so that no additional entry is installed.

7.2. Dynamic demand pairs leaving

In case of a demand pair leaving the network, if the leaving renders the rule to be obsolete, this rule can be safely deleted either by the controller or idle timeout [11]. However, depending on the network traffic pattern, some unused rules can be kept for a longer time for routing future traffic flows. Details of this problem is out of the scope of the paper [18].

8. Rule placement for statistics gathering

In the previous section, we proposed a scheme to route the traffic in an aggregate manner such that TCAM space consumption is minimized and the performance is guaranteed. However, the OpenFlow controller may need access to collect flow statistics and perform fine-grained control on the individual flow under some circumstances. For example, some flows are mice flows initially and become elephant flows later, and the controller need timely access to the detailed statistics on these flows. OpenFlow supports this



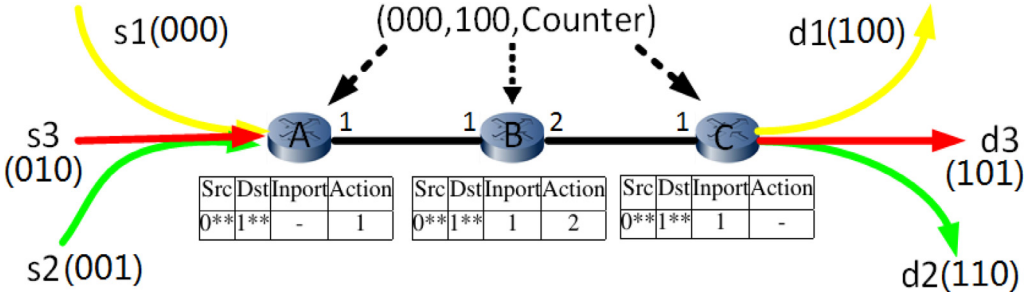
Node	Src	Dst	Inport	Action
A	010	101	-	1
	0**	1**	-	2
B	010	101	1	3
	0**	1**	-	2
C	0**	1**	1	-
D	000	100	*	4
	001	100	*	4
	000	101	*	3
	001	101	*	3
E	010	101	1	-
	0**	1**	2	-

(b)

Node	Src	Dst	Inport	Action
A	0**	1**	-	2
B	0**	1**	-	2
C	0**	1**	1	-
D	010	101	*	3
	000	100	*	4
	001	100	*	4
	000	101	*	3
	001	101	*	3
E	0**	1**	2	-

(c)

Fig. 10. Example of flow tables.



Src	Dst	Inport	Action
0**	1**	-	1

Src	Dst	Inport	Action
0**	1**	1	2

Src	Dst	Inport	Action
0**	1**	1	-

Fig. 11. Example of flow tables.

by involving a counter field in each OpenFlow entry, the counter of the entry will update when the packet matches with the entry, different kinds of counters are supported by OpenFlow, such as number of packets transmitted, number of bytes transmitted, etc [10], the controller will collect the statistics of this flow by querying the data in the counter field. An example is given by Fig. 11. The flows of three demand pairs are transmitted in aggregate manner and the flow tables are also shown in the Fig. 11. To gather the statistics of the flow of demand pair [000, 100], a rule with the source and destination addresses 000 and 100 must be installed, and the controller is free to choose which switch along the path *A, B, C* this rule is inserted since the traffic of [000, 100] will pass through all these three switches.

8.1. Problem formulation

Unlike the objective function defined by (3), in this scenario we are more interested in making sure that all the switches have space to install the rules since failure to install the rule will cause the controller to lose control on the specific flow. For the misbehaved flows (e.g. elephant flow) which consume the majority of the resources, it is necessary for the controller to gather statistics and perform fine-grained control timely. Consider the example of Fig. 11, assume the switch $w(B)$ is lower than $w(A)$ and $w(C)$, to minimize the total cost defined by (3) all the rules will be installed on B until reaching the TCAM space limit of B. Later if there is a need to install rules to collect statistics on the other demand pairs whose only intermediate switch is B (e.g the flow only passes through B), then this rule will be discarded since B is already full. Therefore, instead of Eq. (3), we should minimize the maximum consumption on TCAM space in each switch, that is, minimize the maximum number of rules installed on each switch. Let k_{fine} denote the set of demand pairs which are needed to collect statistics. z_{vk} is a binary variable, $z_{vk} = 1$ indicates that the rule is installed on switch v to gather the statistics for k. q_v indicates the initial number of rules installed on switch v, and $H(k)$ denotes the path carrying the traffic of demand pair k. Then the problem is shown below:

$$\begin{aligned} \underset{z_{vk}\in\{0,1\}}{\text{minimize}} \quad & \lambda \\ \text{subject to} \quad & \sum_{v\in H(k)} z_{vk} = 1, \ \forall k \in K_{fine} \\ & q_v + \sum_{k\in K_{fine}} z_{vk} \le \lambda, \ \forall v \in V \end{aligned}$$

The first constraint ensures that the rule for k is installed on one of the switch along the path $H(k)$, and the second constraint ensures that the total number of rules on each switch is less or equal than λ. We call this problem *Rule Placement Problem (RPP)*, and *RPP* is a NP-hard problem.

8.2. Approximation algorithm of RPP

Next we proposed a 2-approximation algorithm for *RPP*. First we define a new variable ϕ_{vk}:

$$\phi_{vk} = \begin{cases} \infty & \text{if } v \notin H(k) \\ 1 & \text{if } v \in H(k) \end{cases}$$

Then *RPP* can be redefined as follows:

$$\begin{aligned} \underset{z_{vk}\in\{0,1\}}{\text{minimize}} \quad & \lambda \\ \text{subject to} \quad & \sum_{v\in V} z_{vk} = 1, \ \forall k \in K_{fine} \\ & q_v + \sum_{k\in K_{fine}} \phi_{vk} z_{vk} \le \lambda, \ \forall v \in V \end{aligned}$$

And we cite the result from [27]:

Theorem 3. *Let* $v_{ij} > 0$ *for* $i = 1, \ldots, m$, $j = 1, \ldots, n$, $d_i > 0$ *for* $i = 1, \ldots, m$, *and* $t > 0$. *Let* $A_j(t) = \{i | v_{ij} < t\}$ *and* $B_i(t) = \{j | v_{ij} < t\}$, *if*

the following feasibility problem:

$$\sum_{i \in A_j(t)} x_{ij} = 1,\ j = 1, \ldots, n$$

$$\sum_{j \in B_i(t)} v_{ij} x_{ij} \le d_i,\ i = 1, \ldots, m$$

$$x_{ij} \ge 0,\ \text{for } j \in B_i(t), i = 1, \ldots, n$$

has a solution, then any vertex x′ of this polytope defined by the above feasibility problem can be rounded to a binary integer solution of the following problem in polynomial time.

$$\sum_{i \in A_j(t)} x_{ij} = 1,\ j = 1, \ldots, n$$

$$\sum_{j \in B_i(t)} v_{ij} x_{ij} \le d_i + t,\ i = 1, \ldots, m$$

$$x_{ij} \in \{0, 1\},\ \text{for } j \in B_i(t), i = 1, \ldots, n$$

Then we have to following conclusion:

Theorem 4. *Given the constants q_v and ϕ_{vk}, for any $\lambda > 0$, we can find 2-relaxed decision procedure for the RPP that outputs either: 1. 'no', if there is no feasible solution to achieve this maximum TCAM space consumption λ or 2. z_{vk} which generates a maximum TCAM space consumption at most 2λ.*

Proof. For the linear programming problem given in Theorem 4, set $t = \lambda$, $v_{ij} = \phi_{vk}$, $d_i = \lambda - q_v$, $m = |V|$, $n = |K_{fine}|$ and $x_{ij} = z_{vk}$. By Theorem 3, the problem is either: 1. infeasible. 2. there is a set of integer solution z_{vk} such that $\sum_{k \in K_{fine}} \phi_{vk} z_{vk} \le \lambda - q_v + \lambda = 2\lambda - q_v$. Therefore $q_v + \sum_{k \in K_{fine}} \phi_{vk} z_{vk} \le 2\lambda$ and the maximum TCAM space consumption is at most 2λ. □

Let $\epsilon_{low} = \max_{v \in V}(q_v)$ and $\epsilon_{up} = \max_{v \in V}(q_v) + |K_{fine}|$, then ϵ_{low} and ϵ_{up} are the lower and upper bound of the optimal solution of *RPP*. Then we have the following 2-approximation algorithm of *RPP*.

Theorem 5. *Algorithm 4 is a 2-approximation algorithm of RPP.*

Algorithm 4 Two-approximation algorithm.

1: Set $\epsilon_{low} = \max_{v \in V}(q_v)$ and $\epsilon_{up} = \max_{v \in V}(q_v) + |K_{fine}|$
2: **while** $\epsilon_{low} \neq \epsilon_{up}$ **do**
3: Set $\epsilon = \lfloor \frac{1}{2}(\epsilon_{low} + \epsilon_{up}) \rfloor$
4: Set $\lambda = \epsilon$, apply 2-relaxed decision procedure
5: **if** decision procedure outputs 'no' **then**
6: set $\epsilon_{low} = \epsilon_{low} + 1$
7: **else**
8: set $\epsilon_{up} = \epsilon_{up} - 1$
9: output the binary integer solution found by 2-relaxed decision procedure

Proof. In Algorithm 4, it is easy to see that ϵ_{low} is always the lower bound of the optimal solution of *RPP*. The solution generated by the above algorithm will generate a output which is less or equal to $2 \times \epsilon_{low}$, which is less than two time of the optimal solution of *RPP*. □

It is easy to see that the above algorithm will run in polynomial time. Since the difference between ϵ_{up} and ϵ_{low} is cut by half after each iteration. So the 2-relaxed decision procedure is called at most $\mathcal{O}(log|K_{fine}|)$ times. Since the 2-relaxed decision procedure runs in polynomial time by Theorem 4, hence Algorithm 4 will run in polynomial time.

9. Simulations

9.1. Network settings

We evaluated *DSA* on 4 different network topologies, one is a WAN model generated by *GT-ITM* [19], which simulates WANs using Transit-Stub topologies. This network has 100 nodes and 127 undirected links. The other network topologies includes the *Abilene* (11 nodes, 13 undirected links), *Fat Tree* (4 pods, 4 core switch, 52 nodes and 64 undirected links) and *Sprint* (52 nodes, 168 undirected links). The traffic distribution for Abilene and Sprint are available in [20]. We use two models proposed in [20]: *Lognormal* distribution ($\mu = 15.45, \delta = 0.885$), and *Weibull* distribution ($a = 1.87 \times 10^5, b = 0.69$) to model the traffic distribution in the Sprint Network. And we use the *Lognormal* distribution ($\mu = 16.6, \delta = 1.04$) to model the traffic distribution in the Abilene Network. For the *GT-ITM* and *Fat Tree*, we use the Bimodal distribution (generated by mixture of two Gaussian Distributions) proposed in [21]. The Bimodal distribution is proposed based on the observation that only a small fraction of Source-Destination pairs has large flows. Assume each switch has a capacity between 300 – 500 entries. We use the method proposed in [22] to model the link capacity, which claims that the link capacity distribution follows the *Zipf's Law*, and the links whose end nodes with higher degree tend to have larger link capacity. For the purpose of simulation, we set the link capacity to 39.8 Gbps (the transmission rate of optical carrier *OC*768) if the degrees of both endpoints of that link are larger than 3, set the link capacity to 9953.28 Mbps (*OC*192) if one endpoint has degree larger than 3 and degree of the other end point is less or equal 3, set the link capacity to 2.49 Gbps (*OC*48) if the degree of both endpoints is less or equal than 3.

We randomly generate demand pairs that correspond to a source machine and destination machine in the network. Each machine has been assigned a random type B IP address and are connected to a switch in the network. The bandwidth consumption of the flows follows the distributions described above. Since TCAM space aware routing has not been investigated before, and there is no existing routing algorithm that aims to reduce the routing table size, we compare *DSA* with two benchmark routing schemes: *ECMP* and *Valiant Load Balancing (VLB)* which are widely used to achieve load balancing on link resources. Our purpose is to demonstrate that *DSA* can substantially reduce of TCAM space without sacrificing too much in terms of load balancing of link resources. *ECMP* is a routing strategy which works by splitting the traffic equally over the multiple paths with the same length (number of hops) [23]. In *VLB*, the flows of the same demand pair are first sent to some intermediate nodes, then forwarded to the destination [24].

After the paths are calculated by the two benchmark routing schemes, the corresponding rules (s_k, i, d_k, j) are installed to direct the flows. All the rules contain fully specified addresses s_k and d_k so that they can not be reused by the other flows. We run each algorithm 100 times and take the average results. All the evaluation is run on a machine with 8 GB of RAM and Quad-Core Intel i7 CPU(3.2 GHz). For the evaluation, we set the weight of each switch in Eq. (3) to 1, therefore the total cost generated by Eq. (1) equals the total number of entries installed. We compare performance of the algorithm using a metric called Traffic Saving Ratio. Assume the total amount of TCAM space consumed by *DSA* is T_p, and total amount of TCAM space consumed by the benchmark algorithm is T_b, then *Traffic Saving Ratio (TSP)* is defined as:

$$TSP = (T_b - T_p)/T_b \tag{11}$$

9.2. Evaluation of the TSP

First we evaluate the relation between the number of demand pairs and *TSP*. We do not limit the maximum routing group size.

Table 2
Mean of TSP on all the network topologies.

Network	Number of flow		60	70	80	90	100
Abilene	Lognormal	TSP_1	0.4264	0.6331	0.7176	0.7518	0.7851
		TSP_2	0.4051	0.6318	0.7015	0.7427	0.7881
Network	Number of flow		80	120	160	200	240
Sprint	Weibull	TSP_1	0.2570	0.3740	0.5252	0.5805	0.6659
		TSP_2	0.2551	0.4553	0.5038	0.6519	0.7718
	Lognormal	TSP_1	0.2041	0.2869	0.5437	0.5809	0.6003
		TSP_2	0.2473	0.3933	0.5154	0.6715	0.7258
Network	Number of flow		200	300	400	500	600
GT-ITM	Bimodal	TSP_1	0.4253	0.4353	0.5766	0.6107	0.6611
		TSP_2	0.4542	0.4604	0.5599	0.6034	0.7912
Network	Number of flow		100	150	200	250	300
Fat tree	Bimodal	TSP_1	0.1981	0.2986	0.4334	0.6008	0.6745
		TSP_2	0.2158	0.2974	0.4298	0.6177	0.7208

Table 3
90% confidence interval of TSP on all the network topologies.

Network	Number of flow		60	70	80	90	100
Abilene	Lognormal	TSP_1	0.3988-0.4511	0.6003-0.6289	0.6901-0.7372	0.7314-0.77	0.7701-0.8013
		TSP_2	0.3537-0.4451	0.5924-0.6601	0.6705-0.7347	0.7223-0.7664	0.7606-0.8123
Network	Number of flow		80	120	160	200	240
Sprint	Weibull	TSP_1	0.2261-0.2810	0.3437-0.3999	0.5004-0.5414	0.5605-0.6071	0.6347-0.6911
		TSP_2	0.2227-0.2796	0.4257-0.4829	0.4679-0.5339	0.6307-0.6733	0.7410-0.7997
	Lognormal	TSP_1	0.1818-0.2301	0.2598-0.3044	0.5179-0.5771	0.5588-0.6002	0.5888-0.6268
		TSP_2	0.2186-0.2735	0.3655-0.4219	0.4884-0.5400	0.6550-0.6943	0.6998-0.7445
Network	Number of flow		200	300	400	500	600
GT-ITM	Bimodal	TSP_1	0.3979-0.4515	0.4003-0.4668	0.5501-0.5995	0.5911-0.6386	0.6332-0.6904
		TSP_2	0.4222-0.4824	0.4298-0.4911	0.5345-0.5885	0.5799-0.6303	0.7649-0.8219
Network	Number of flow		100	150	200	250	300
Fat tree	Bimodal	TSP_1	0.1771-0.2175	0.2687-0.3279	0.4040-0.4655	0.5785-0.6233	0.6550-0.6991
		TSP_2	0.1965-0.2379	0.2688-0.3240	0.4030-0.4561	0.5888-0.6462	0.6975-0.7478

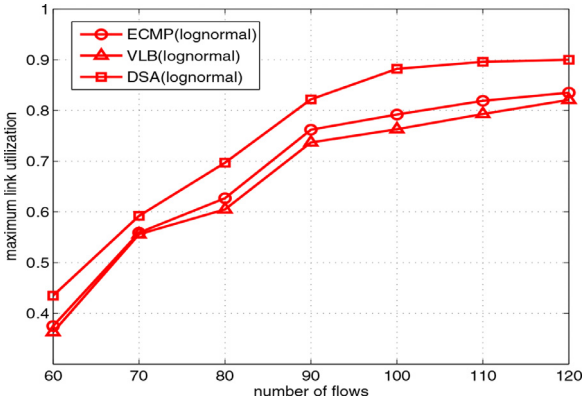


Fig. 12. Link utilization (Abilene).

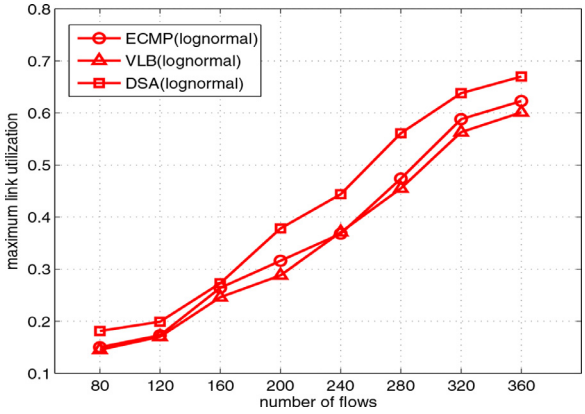


Fig. 13. Link utilization (Sprint).

Table 2 shows the relations between the number of flows and mean of *TSP* with different networks and different traffic distributions and Table 3 shows the 90% confidence intervals of *TSP*. TSP_1 is the *TSP* of the *ECMP* and TSP_2 is the *TSP* of the *VLB*. The *DSA* can achieve 20% – 80% saving on the TCAM space with different network topologies and traffic distributions. The saving also grows with the number of flows. This is because as the number of flows increases, more flows can be aggregated for saving TCAM space. Moreover, if we neglect the first packet of each flow which is forwarded to the controller, the TCAM space saving almost equals to the saving in the number of control traffic between the controller and the OpenFlow switches. This is because each entry in the switches requires a control packet for installation.

Fig. 12–15 show the relations between the number of flows and the mean of maximum link utilization of different algorithms over different traffic distributions. All of the standard deviations of the

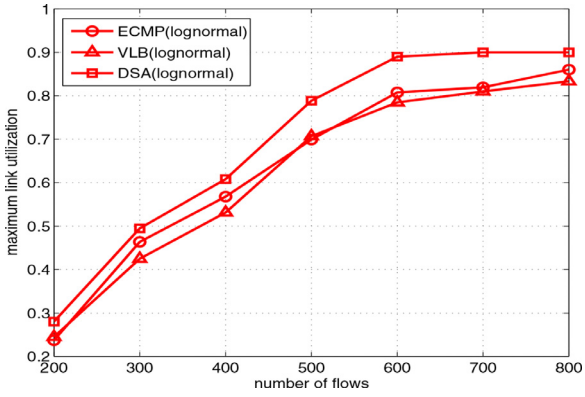


Fig. 14. Link utilization (GT-ITM).

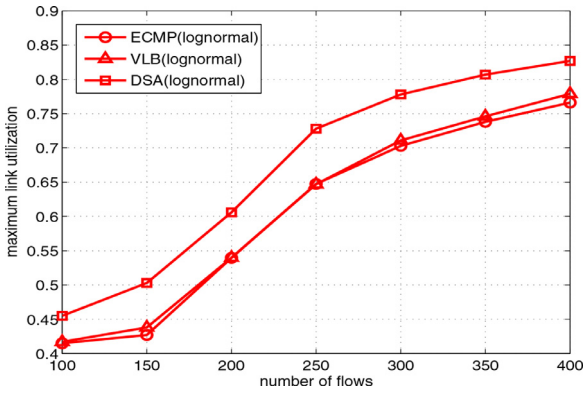


Fig. 15. Link utilization (Fat Tree).

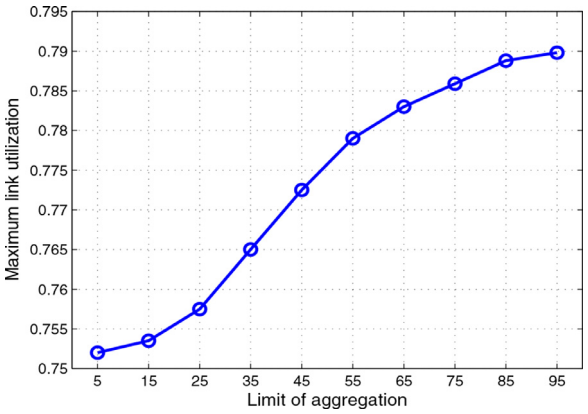


Fig. 16. Change on link utilization.

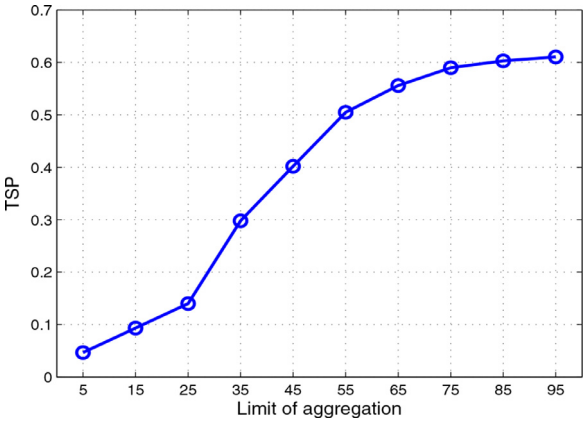


Fig. 17. TSP on GT-ITM.

maximum link utilizations are less than 0.08, When calculating the link utilization of *DSA*, the threshold on link utilization rate, β is set to 0.9. The maximum link utilization rate of *DSA* is on average 10 – 17% higher than that of *ECMP* and *VLB*. Despite the higher link utilization rate of *DSA*, considering the huge savings on the TCAM space, we believe this is a fair trade-off.

As mentioned before, we can tune the value of L to balance the trade-off between TCAM space saving and maximum link utilization, we evaluate the impact of the limit of aggregation L on the *TSP* and maximum link utilization rate on the *GT-ITM* network with 500 demand pairs. As shown in Figs. 16 and 17, when L decreases from 95 to 5, the *TSP* decreases from 0.6107 to 0.04665, the reason is that L affects the size of the routing groups, a small L causes a smaller size of routing groups therefore the degree of flow aggregation decreases. At the same time, the maximum link utilization rate also decreases slightly from 0.79 to 0.752. This is because small routing group leads to fine-grained routes which in turn reduces maximum link utilization.

Table 4
Performance and running time comparison.

Performance	Node TSP	Abilene 0.5979	Sprint 0.5435	Tree 0.4894	GT-ITM 0.4001
Running time	Network DSA	Abilene 0.19 ms	Sprint 0.29 ms	Tree 0.32 ms	GT-ITM 0.37 ms

9.3. Evaluation of the DANA

We generate demand pairs that are attached to some random nodes in the network. Each demand pair has a random type B source and destination IP address, and all the demand pairs are connected by installing the rules generated by DSA. To emulate the dynamic entering of new demand pairs, we generate 50 new demand pairs and run the DANA to add the flows. We compare the performance of DANA with *shortest path algorithm (SPA)*, which routes traffic along shortest paths. All the rules installed for SPA are fully specified addresses. TSP is defined in a similar manner as (11), with T_b and T_p means the total number of rules generated by SPA and DANA to direct the new flows.

Tables 4 and 5 show the means and 90% confidence intervals of performance as well as the running time of the two algorithms. As the tables shows, DANA can achieve 40% – 60% saving on TCAM space. The running time of DANA increases moderately with the network size. But overall running time of the algorithm is still reasonable.

9.4. Evaluation of the two-approximation algorithm for RPP

We reuse the traffic flow routing generated by EPP and ERP in Section 9.2 on the four different network topologies. We also randomly pick some flows that need to be controlled in fine-grain scale, and run the two approximation algorithms that give the rule placement suggestions. We then compare it with the optimal solution generated by exhaustive search method. The total number of flows is 80, and the result generated by two-approximation algorithm is normalized to the optimal solution. We pick 10 and 15 fine-grained controlled flows from the 80 flows, and the simulation results are summarized in Tables 6 and 7.

As it is shown from Table 6, the total cost generated by the two-approximation algorithm is on average 12% – 20% higher than the optimal solution, which demonstrates the efficiency of the two-approximation algorithm.

10. Testbed deployment

We evaluated the functionality and implementability of the DSA in a real testbed. We built an overlay network that follows Abilene network topology by using a software switch (OpenVswitch) running on virtual machines. The OpenVswitches communicate with each other by using the virtual extensible LAN. The centralized controller (Ryu) can configure the entries in the switches to build the routing paths. We built three source VMs and three destination VMs (three demand pairs), each VM is assigned an IP address. The routing module on top of the Ryu controller takes the connection demands as the input and sends the results of DSA to the implementing module which installs the relative OpenFlow rules on the switches (Fig. 18). For comparison, we also used the shortest path algorithm (Dijkstra's Algorithm) to connect the demands pairs. For DSA, a total 14 entries are installed on the switches, and

Table 5
90% confidence interval of TSP.

Performance	Node	Abilene	Sprint	Tree	GT-ITM
	TSP	0.5808-0.6102	0.5288-0.5660	0.4606-0.5000	0.3768-0.4242

Table 6
Performance of two-approximation algorithm.

Mean of the ratio (10 fine grained flows)	Abilene 1.1771	Sprint 1.1808	Tree 1.1345	GT-ITM 1.1447
Mean of the ratio (15 fine grained flows)	Abilene 1.1680	Sprint 1.2018	Tree 1.1677	GT-ITM 1.1494

Table 7
Performance of two-approximation algorithm.

90% CI of the ratio(10 fine grained flows)	Abilene 1.1212-1.2016	Sprint 1.1118-1.259	Tree 1.1004-1.2280	GT-ITM 1.0642-1.2288
90% CI of the ratio (15 fine grained flows)	Abilene 1.1163-1.2089	Sprint 1.1077-1.2930	Tree 1.0969-1.2335	GT-ITM 1.0612-1.2307

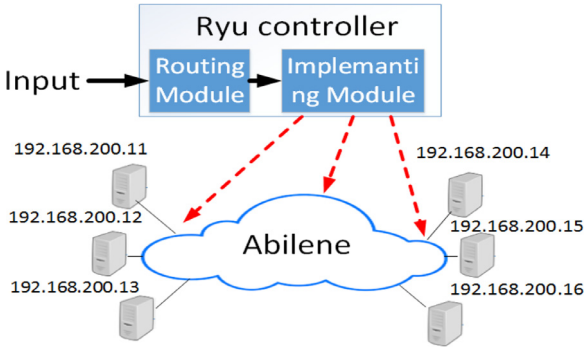


Fig. 18. Real testbed experiment.

the total time taken for building the path is 0.028 s. For Dijkstra's Algorithm, total 30 entries are installed on the switches with the total time 0.061 s. Hence, DSA clearly saves TCAM space and path set up time.

11. Conclusions

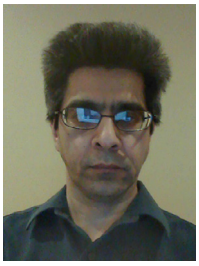
In this paper, we proposed an efficient routing scheme to achieve savings on TCAM space in SDN without causing network congestions. We provide algorithms for both the static and dynamic scenarios. Moreover, for the purpose of statistics gathering on the flow entries, we also propose a rule placement algorithm to achieve load balancing on TCAM space. Experiments show that the proposed routing scheme can achieve 20% – 80% saving on TCAM space with 10% – 17% increase in maximum link utilization. Finally, a preliminary version of the DSA has been implemented on the real testbed environment.

References

[1] N. Kang, Z. Liu, J. Rexford, D. Walker, Optimizing the 'one big switch' abstraction in software-defined networks, in: Proceedings of ACM Conext, 2013.
[2] C. Hong, S. Kandula, R. Mahajan, et al., Achieving high utilization with software-driven WAN, in: Proceedings of ACM Sigcomm, 2013.
[3] M. Zhang, et al., GreenTE: power-aware traffic engineering, in: Proceedings of IEEE ICNP, 2013.
[4] E. Oki, et al., Fine two-phase routing with traffic matrix, in: Proceedings of IEEE ICCCN, 2009.
[5] X. Liu, R. Meiners, E. Torng, TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs, IEEE ACM Trans. Netw. 18 (2) (APRIL 2010).
[6] A.R. Curtis, J.C. Mogul, J. Tourrilhes, et al., Devoflow: Scaling flow management for high-performance networks, in: the Proceedings of ACM Sigcomm, 2011.
[7] P. Lekkas, Network Processors : Architectures, Protocols and Platforms, McGraw Hill Professional, 2003. Jul 28
[8] P.T. Congdon, et al., Simultaneously reducing latency and power consumption in openflow switches, IEEE ACM Trans. Netw. 22.3 (2014) 1007–1020.
[9] Y. Kanizo, D. Hay, I. Keslassy, Palette: Distributing tables in software-defined networks, in: Proceedings of IEEE Infocom, 2013.
[10] M. Alizadeh, A. Greenberg, et al., DCTCP:efficient packet transport for the commoditized data center, in: Proceedings of ACM Sigcomm, 2010.
[11] Openflow Spec, www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.
[12] M. Moshref, M. Yu, A. Sharma, R. Govindan, Scalable rule management for data centers, in: Proceedings of USENIX NSDI, 2013.
[13] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, in: Proceedings of Sigcomm, 2010.
[14] S. Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of ACM HotSDN, 2012.
[15] A.S. Tam, et al., Use of Devolved Controllers in Data Center Networks, Infocom Computer Communications Workshops, 2011.
[16] K. Phemius, M. Bouet, J. Leguay, DISCO: Distributed multi-domain SDN controllers, in: Proceedings of IEEE NOMS, 2014.
[17] D.P. Williamson, D.B. Shmoys, The design of approximation algorithms, Cambridge university press, 2011. http://www.designofapproxalgs.com/.
[18] H. Zhu, H. Fan, X. Luo, Y. Jin, Intelligent timeout master: Dynamic timeout for SDN-based data centers, in: Proceedings of IEEE IM, 2015.
[19] GT-ITM website: www.cc.gatech.edu/projects/gtitm/.
[20] A. Nucci, et al., The problem of synthetically generating IP traffic matrices: initial recommendations, ACM Sigcomm CCR, July 2005.
[21] A. Medina, N. Taft, et al., Traffic matrix estimation: existing techniques and new directions, in: Proceedings of Sigcomm, 2002.
[22] T. Hirayama, S. Arakawa, S. Hosoki, M. Murata, Models of link capacity distribution in ISP's router-level topologies, JCNC, 2011.
[23] C. Hopps, Analysis of an equal-cost multi-path algorithm, (2000).
[24] R. Zhang-Shen, Valiant Load-Balancing: Building Networks That Can Support All Traffic Matrices, Springer, 2010.
[25] R. Cohen, L. Lewin-Eytan, J. Naor, D. Raz, On the effect of forwarding table size on SDN network utilization, in: Proceedings of Infocom, 2014.
[26] N. Katta, et al., Infinite cacheflow in software-defined networks, in: Proceedings of HotSDN, 2014.
[27] L.J. Karel, D. Shmoys, E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, in: Proceedings of 28th Annual Symposium on Foundations of Computer Science, 1987.

Sai Qian Zhang received his B.A.Sc and M.A.Sc degree in Electrical engineering from University of Toronto, Canada, in 2013 and 2016 respectively. He is currently pursuing his doctoral degree at Harvard university. His research interest includes traffic engineering, routing algorithms, software defined networking, network function virtualization, etc.

Qi Zhang received his Ph. D, M. Sc and B. A. Sc. from University of Waterloo (Canada), Queen's University (Canada) and University of Ottawa (Canada), respectively. He is currently a post-doctoral fellow in the Department of Electrical and Computer Engineering at University of Toronto (Canada). His research focuses on resource management for Cloud data centers and applications. He is also interested in related research areas including network and enterprise service management.



Ali Tizghadam (ali.tizghadam@utoronto.ca) is a Senior Research Associate in the ECE Department at the University of Toronto. He is leading the development CVST research project (Connected Vehicles & Smart transportation), which is an ORF Funded university-industry-government partnership in Canada to build a flexible and open transportation application platform for research and business purposes. He received his M.A.Sc. and Ph.D. in electrical and computer engineering from the University of Tehran (1994) and University of Toronto (2009), respectively. Between his M.A.Sc. and Ph.D. studies he worked in the industry for about 10 years, where he gained an abundance of experience in different aspects of networking from telecommunication networks to transport systems and power grids. His major research interests are in applications of network science in transportation, communications, green networking, autonomic network control and management, network optimization, and smart grid.

Byungchul Park is a post-doctoral researcher at the University of Toronto. He received his Ph.D. (2012) and B.Sc. (2006) degree in computer science from POSTECH, Korea. His research interests include Internet traffic measurement and analysis.

Hadi Bannazadeh holds a PhD from the University of Toronto Department of Electrical & Computer Engineering. After graduating, he worked at Cisco Systems as a Senior Network Software Engineer. In 2011, he returned to the University of Toronto to lead the efforts towards the creation of Canadian national testbed as part of the Smart Applications on Virtual Infrastructure (SAVI) research project. Since then, he has been the Chief Testbed Architect for the SAVI project. Hadi main research interest is in the field of Software Defined Infrastructure (SDI) including Software Defined Networking (SDN) and Cloud Computing.

Raouf Boutaba received the M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris, in 1990 and 1994, respectively. He is currently a professor of computer science and the Associate Dean Research of the Faculty of Mathematics at the University of Waterloo (Canada). His research interests include resource and service management in networks and distributed systems. He is the founding editor in chief of the IEEE Transactions on Network and Service Management (20072010) and on the editorial boards of other journals. He received several best paper awards and recognitions including the Premiers Research Excellence Award, the IEEE ComSoc Hal Sobol, Fred W. Ellersick, Joe LociCero, Dan Stokesbury, Salah Aidarous Awards, and the IEEE Canada McNaughton Gold Medal. He is a fellow of the IEEE, the Engineering Institute of Canada, and the Canadian Academy of Engineering.

Professor Alberto Leon-Garcia is Distinguished Professor in Electrical and Computer Engineering at the University of Toronto. He is a Fellow of the Institute of Electronics and Electrical Engineering "for contributions to multiplexing and switching of integrated services traffic". He is also a Fellow of the Engineering Institute of Canada and the American Association for the Advancement of Science. He has received the 2006 Thomas Eadie Medal from the Royal Society of Canada and the 2010 IEEE Canada A. G. L. McNaughton Gold Medal for his contributions to the area of communications. Professor Leon-Garcia is author of the leading textbooks: Probability and Random Processes for Electrical Engineering, and Communication Networks: Fundamental Concepts and Key Architecture. He is currently Scientific Director of the NSERC Strategic Network for Smart Applications on Virtual Infrastructures.