# Traffic Engineering to Minimize The Number of Rules in SDN Datacenters

Rajorshi Biswas and Jie Wu
Department of Computer and Information Sciences
Temple University, Philadelphia, PA, USA
{rajorshi, jiewu}@temple.edu

***Abstract*—Traffic engineering is one of the important parts in a datacenter. Software defined networking (SDN) opens opportunities for easier traffic engineering. Functionalities of an SDN switch are determined by the forwarding rules installed on it. Because of the limited space on SDN switches, it is important to keep the number of rules as small as possible. If the number of rules in a switch is higher than a threshold, then the forwarding delay jumps up. A virtual tunnel based approach helps to reduce the number of rules in an SDN switch while preventing link flooding attack vulnerability. A virtual tunnel is basically a conceptual structure in the controller of an SDN network where a group of flows follow a common path. A wise formulation of tunnels can reduce the number of rules needed dramatically by forwarding multiple flows through the same tunnels with the tunnel's common rules. In this paper, we address this important issue and propose a mechanism to keep the number of rules minimum. We formulate two problems and propose clustering-based and greedy solutions with an approximation ratio. Our first problem is to build some tunnels using the minimum number of edges and rules. Our second problem considers some predetermined tunnels and finds an ID assignment to the flows so that the number of rules needed to forward them is minimum. We conduct extensive simulations and experiments on our datacenter to validate our proposed model.**

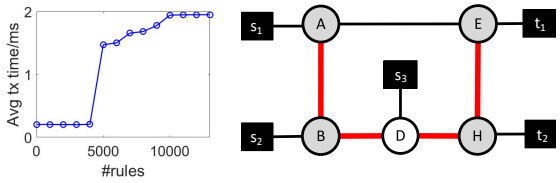
***Index Terms*—Traffic engineering, link flooding attack, software defined networking, minimize rules***

## I. Introduction

Nowadays, software defined networking (SDN) switches are being used instead of regular routers in datacenters. In the SDN architecture, a centralised software, called a controller, controls all of the routing of flows. The controller can get the global view of the topology including link status, flow rates and other properties. This centralised architecture has made flow management easy. In addition to the limited capacity of links, routers, or switches, there is another limitation on number of rules in SDN switches.

To observe the importance of the number of rules, we conduct an experiment on a Pica8 P-3297 [1] SDN switch. We insert different numbers of rules in the SDN switch and observe the packet transmission delay between two machines. The machines are directly connected to the SDN switch which means they are two hops away from each other. If the number of rules is greater than $4,000$, then the delay increases suddenly by 1 ms. After that, transmission delay increases with the number of rules. This experiment motivates us to conduct research on minimizing the number of rules.

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128.




(a) Tunnel creation. (b) Flow redirection.

Fig. 1: An introductory example.

One way to minimize the number of rules is to group the flows and use a common rule to forward them. It is always beneficiary to aggregate flows that travels a long common path which can be achieved by creating some virtual tunnels. The virtual tunnels are not similar to TLS or VPN. These tunnels are only conceptual in the SDN controller. Packets of a flow are tagged with a tunnel ID if they travel through a tunnel. The most challenging parts are creating the tunnels and assigning IDs to the channels so that the number of rules is minimal. Creating tunnels needs to maintain the capacity constraints of the links. Vulnerability to link flooding attack is another important issue to take in to account while creating tunnels. If a link is highly utilised, then it may be a lucrative spot to attackers and they can be overwhelmed by very small amount of attack traffic.

For example, in Fig 1(b), there are three flows $f_1 = s_1 \rightarrow t_2$, $f_2 = s_2 \rightarrow t_2$, and $f_3 = s_3 \rightarrow t_1$. First, we want to forward the flows through the shortest path or with the minimum number of rules. $f_1$ is forwarded through path $\{A, E, H\}$ and three rules are needed. $f_2$ is forwarded through path $\{B, D, H\}$ and three rules are needed. Similarly, for $f_3$ we need three more rules. Therefore, in total we need nine number of rules. Second, we want to use the tunneling approach. Let, we create a tunnel $\{A, B, D, H, E\}$. In this situation, we need four rules to create the tunnel (forwarding rules at node A,B, D, and H). We need additional three rules to take exit from the tunnel (deliver packets to $t_2$ and $t_1$). Therefore, we need seven rules only if we use this tunnel. The problem with this tunnel is highly utilized link $(D, H)$ which is vulnerable to link flooding attack. To avoid any link to be vulnerable to attack we create a tunnel $\{B, D, H, E\}$ and forward $f_2$ and $f_3$ through the tunnel. We need three rules for the tunnel and two to take exit from the tunnel. In addition to this, we need three rules to forward $f_1$ through $\{A, E, H\}$. Therefore, in total we need eight number of rules, which higher than with vulnerability but lower than shortest path routing. In reality,

the number of rules in a switch is much higher and we can save a large number of rules using the tunneling approach.

In this paper, we utilize the possibility of tunnel creation to minimize the number of rules. We formulate a problem to find the tunnels and we propose a clustering based solution. Additionally, we formulate another problem to assign ID of the tunnels so that the number of rules needed to forward the packets is minimum. We provide an optimal solution for a simple tunnel structure. We provide a greedy set cover based decomposition of complex structure and apply the solution to the simple structure. We also provide the approximation ratio of our proposed solution. Therefore, our main contributions are the following:

1) We formulate a problem to create tunnels that minimize the number of rules. We provide a clustering based solution to this problem.
2) We study another problem to assign tunnel IDs in such a way that minimizes the number of rules. We provide a set cover based solution to the problem and provide approximation ratio of our solution.
3) We conduct extensive simulations and the experimental results support our model.

The remainder of this paper is arranged as follows. Section II presents some related work and their limitations. In Section III, we present the system model. In Section IV, we present the problem of tunnel creation and a clustering based solution. Section V contains the problem of assigning IDs to the tunnels and solutions. Section VI and Section VII present the simulation and experimental and results. Finally, Section VIII concludes our paper.

## II. Related Work

Unlike networks with regular topology [2] or networks with fault-tolerant topology [3] where re-rerouting under attacks and failures can be done using inherit topology redundancy and structure, in networks with irregular topology, there are three types of works defending LFA and link congestion. Firstly, there statistical methods, including correlation, entropy, covariance, divergence, cross-correlation, and information gain are used to detect attack traffic [4]. Other types of congestion mitigation system includes multi-path routing [5–8]. These schemes either do not consider SDN switches or the number of rules or SDN switch capabilities.

Secondly, there exists some linear programming based solutions in SDN environments. In [9], the authors considers multiple failures of links and multi-flows rerouting. They propose a model for reducing communication overhead between controller and switch during flow rerouting to minimize flow rules. They formulate the problem as a $0-1$ nonlinear programming model and solve using Lagrange relaxation. An approach to ensure traffic reachability for a single link failuire is proposed in [10]. They redirect traffic on the failed link to SDN switches via some pre-configured IP tunnels. They can configure multiple backup paths using SDN switches which allows the system to mitigate congestion very fast. In [11], authors minimize flow rules for rerouting multi-flows when there are multiple link failures in the network. They propose a model to metric the communication overhead between controller and switch during flow rerouting and we formulate a 0-1 nonlinear programming model. Their model can avoid link congestion in the network and can provide the minimum flow rules. In [12], authors consider minimizing the number of forwarding rules installed on SDN switches. They introduce an heuristic algorithm that minimizes rules that are only used by network statistics analyzer applications. In [13], authors formulate a problem to minimize the number of rules in multicast routing model for multiple multicast requests as an integer linear programming (ILP) problem and solve using traditional solvers. In [14], authors also propose an ILP to minimize the overall delay due to flow table space and transmission overhead. A flow forwarding scheme called PASR is prposed in [15]. The PASR can learn the flow path information and implement flow aggregation. They also present an intelligent encoding algorithm to minimize the number of rules. In [16], authors propose a destination-based routing solution using reinforcement learning and linear programming to reduce the number of rules. It forwards majority of the traffic on equal-cost multi-path and redistributes some traffic using reinforcement learning algorithm.

Finally, in [17], authors propose a hybrid and scalable SDN datacenter using both wireless and wired connections. They formulate a routing minimization problem to minimize the total number of installed rules in the switches. They propose an heuristic algorithm to find a routing with rule aggregation. In [18], minimize flow rules for rerouting multi-flows when network occurs multiple link failures. They propose a model to measure the communication overhead between controller and switch during flow rerouting. They formulate the problem as a 0-1 linear programming and solve the model using decomposition based on Lagrange relaxation. Some other works such as [19], considers the problem of minimizing the number of rules in SDN switches by analyzing network statistics. They propose an heuristic algorithm that creates a reduced representation of rules in the SDN switches in network. They do not consider link congestion or an attack scenario. Therefore, a new method that considers the link congestion, aggregation of flows, and using minimum number of rules is necessary.

## III. System Model

### A. Network Model

Our network is composed of gateway nodes, SDN switches, sources, destinations, and flows. We assume that the controller knows the links, their usage, and SDN switches. Therefore, the controller has the global view of the topology. A rule in an SDN switch is a packet forwarding policy. The properties in the header of an incoming packet are matched with some criteria (e.g. source address, destination address, incoming port, protocol) of the rules. If a packet is matched with criteria of any of the rules, then actions are taken according to that rule. For simplicity, we consider the destination address as the only matching criteria of the rules of the regular flow.

We introduce another criteria for matching called tunnel ID. The rules that have tunnel ID in their matching criteria are called tunnel rules. A tunnel ID is assigned to a packet of incoming flows by the gateway nodes. Assigning the tunnel ID means attaching a specific field in the header of the incoming packet. Once a tunnel ID is attached to a packet, it is forwarded towards the tunnel first. After it reaches tunnel, it is forwarded through the tunnel specified by the ID towards the destination using the tunnel rules. Once the packet is at the closest node from the destination, it exits the tunnel. We call the rule responsible for exiting a tunnel an exit rule. Once the packet exits the tunnel, it is forwarded to the destination using regular rules. A tunnel in a datacenter is a virtual concept only maintain by the SDN controller. The controller creates the virtual tunnels in such a way that it takes a small number of rules to forward all the flows. The creation of tunnels is a dynamic process; when the topology changes, the controller may change the structure of the existing tunnels. If a link on some tunnels goes down, the controller can recalculate the tunnels or discard the tunnels. The controller can add, modify, or delete rules from an SDN switch. After creating the tunnels it installs the corresponding tunnel rules to forward the packets having the tunnel ID attached.

There is limited space to store rules from where the switch can lookup them fast. If this limit is exhausted by some rules, then additional rules are stored in slow memory; therefore the number of rules in an SDN switch is very important. The disadvantage of fast memory is it takes a high amount of power and the power is proportional to the number of stored rules. The advantage of using slow memory for storing rules is that they consume less energy. Therefore, when the fast memory is full, the rule matching takes a long time and packet forwarding delays. A common issue in a datacenter is the network congestion due to regular traffic and attack traffic. Detecting the attack traffic by analysing the packets is a slow process. Sometimes it needs deep packet inspection and large number of data processing. The detection and blockage of attack traffic is out of the scope of this paper and we focus on forwarding some traffic in a way that use a few number of rules considering the network congestion issues.

An attacker can congest links by analyzing the topology and the routing of the network. It selects one or multiple links as a target links and flood them by sending attack packets. Target links are selected based on the number of flows through it and the remaining capacity of the link. The selected target links should forward a large number of flows so that a large number of flows can be be affected. The remaining capacity of bandwidth must be lower or equal to the attacker's capacity. In fact, a link with low remaining capacity is attractive to the attacker because it can be overwhelmed with a few amount of attack traffic. After selecting the target link, the attacker selects the decoy server and bot pairs to generate traffic. Decoy servers are used to receive the attack traffic from the bots and they are owned by the attacker. The bots are malicious programs residing in users' computers which is capable of generating unnecessary traffic to any destinations. The selection of a pair

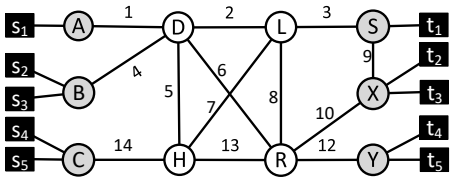


Fig. 2: An example.

$< bot, decoy\ server >$ is done in such a way that the traffic passes through one of the targeted link. An attack flow cannot congest a link, but when a large number of flows pass through a link it becomes overwhelmed. As a result, the regular traffic that passes through the target links suffer from packet drop and a low data rate.

## IV. Tunnel Creation

### A. Problem I Definition

***Problem I:*** *Find virtual tunnels to forward the flows so that the total number of used links is minimized while ensuring the link capacity constraint.*

In this problem, we are given the topology and the flows and we minimize the total number of links used for forwarding the traffic. This is because there is a direct relation between the number of used links and the number of rules. Let us assume that there are $K$ flows $\{f_1, f_2, ..., f_K\}$. A flow is defined by three tuples $f_i = (s_i, t_i, r_i)$, where $s_i$, $t_i$, and $r_i$ are the gateway node of the source, destination, and data-rate of flow $f_i$, respectively. Let, $f_i(u, v)$ denote whether flow $f_i$ travels through link $(u, v)$. $f_i(u, v) = 0$ (or 1) means flow $f_i$ travels (or does not travel) through link $(u, v)$. We will ommit the subscript $i$ if any of the flow travels through $(u, v)$. Therefore, the problem can be expressed as the following:

$$\begin{aligned} &\text{minimize} \sum_{(u,v)\in E} f(u,v) \\ &\text{subject to} \quad \forall_{(u,v)\in E} \sum_{i=1}^{K} f_i(u,v) \times r_i \leq C(u,v) \\ &\qquad\qquad \forall_{1\leq i\leq K} \exists_{p_i} \forall_{(u,v)\in p_i} f_i(u,v) = 1 \end{aligned} \tag{1}$$

Here, $C(u, v)$ denotes the capacity of link $(u, v)$ and $p_i$ denotes the path from $s_i$ to $t_i$. The first constraint indicates that the data rate of the flows going through any link cannot be greater that the capacity of the link. The second constraint ensures connectivity between the source and destination gateway of the flows.

### B. A Grouping-based Solution

The problem is NP-Hard and we use hierarchical and k-means clustering to find a solution. The problem is similar to the steiner tree problem, where the terminal vertices are the sources and destinations of the flows. If we consider the weight of each link to be the same, the steiner tree contains the minimum number of links. Since we are using the clustering techniques, there is no approximation ratio of this solution. We first group the gateway nodes based on the shortest path distance among them. We use a hierarchical clustering approach to group the nodes. Parameters of hierarchical clustering are

set to get a desired number of groups. After getting the desired groups, we discard the intra-group flows. This is because the distance between the source and destination gateway nodes much are smaller than that of the inter-group flows. It is not beneficiary to create or use tunnels for a flow that can travel short distance. The intra-group flows will be forwarded according to the existing forwarding rules. Now the problem is basically a min cost multi-commodity flow problem and the solutions are linear programming based. We propose a greedy solution that is fast without a performance guarantee.

Next, we describe the proposed greedy solution. We first pick a couple of groups and create tunnels between them. To create the tunnels, we find routing paths of the flows using the minimum number of links. Then, we pick a flow between the groups and compute a shortest path routing from the source to destination gateway nodes. The edges on the paths are added to the bucket of picked-up edges. Then we pick another flow and it is routed in the way that adds minimum number of new edges to the picked-up edges. We maintain the capacity constraint when we pick up an edge for routing new flows. After assigning all paths to all of the flows, we get the final set of picked-up edges that will form the tunnels. The number of links to forward the flows has a great impact on the number of rules. Usually, to forward a flow through a link we need a rule at the egress SDN switch. Some rules forward multiple rules but in general, a less number of links need less number of rules.

After that, we create the tunnels using the links we picked for each flow using the k-means clustering technique. The links are considered as features and the clusters are considered as tunnels. We select the number of clusters that produce the least number of rules.

$$R(\kappa) = \sum_{k=1}^{\kappa} \sum_{i=1}^{K} |P(f_i) - C_k| + |C_k| \tag{2}$$

Here, $P(f_i)$ is the set of links used to forward $f_i$. $C_k$ is centroid of the $k$th cluster which means the set of links on the tunnel. The first part of the equation is for the links that are not on the tunnel. We need a special forwarding rule for each such link. The second part is the length of the tunnel because we need a rule for each link in the tunnel. The special forwarding rules get priority over the tunnel forwarding rules. There is a trade-off between the number of rules and the value of $K$. When $\kappa = 1$, almost all of the flows will be forwarded using special rules. Therefore, the number of rules is very high. When $\kappa = K$, then each flow will be forwarded through a tunnel, then the total number of rules for tunnels will also be high. When $\kappa$ is in between, then the number of special rules and the number of tunnels are both small. Therefore, the total number of rules will also be small.

### C. An Example

Let us consider the example in Fig. 2. There are seven flows which are listed in Table I. We group the gateways using hierarchical clustering and get two groups, $\{A, B, C\}$ and $\{S, X, Y\}$. As there is only a couple of groups, we need to create tunnels between them. Let us assume that each link can forward four flows simultaneously. According to the algorithm, we pick flow $f_1$ and forward it through a path that uses a smaller number of links. Therefore, $f_1$ is forwarded using links $\{1, 2, 3\}$. Then, we pick $f_2$ and at least one new edge (9) is needed to forward it through $\{1, 2, 3, 9\}$. After that we pick up $f_3$ followed by $f_4$. Both flows are forwarded via $\{1, 6, 12\}$. Similarly, $f_5$ and $f_6$ are picked up and forwarded via $\{14, 8, 3\}$. Now, we pick $f_7$ and can forward it through $\{4, 2, 3, 9\}$ with the minimum new edges. Because of the exhausted link 4, we cannot use this path. Therefore, we choose $\{4, 6, 10\}$ to forward $f_7$ and $f_8$.

**Algorithm 1** Find tunnels

**Input:** Topology $G(V, E)$, set of flows $F$.
**Output:** A set of tunnels $T$.

```
Procedure: FIND-TUNNELS(G, F)
    f ← REMOVE(F)
    E ← ∅,   ∀_{1≤i≤|N|} P_i ← ∅
    while F ≠ ∅ do
        P_i ← {p : ∀p, p' |p − E| <= p'}
        E ← E ∪ P_i
        f ← REMOVE(F)
    Min ← ∞
    for k = 1 to |N| do
        C ← K-MEANS(P, K)
        r ← Σ_{k'=1}^{k} Σ_{i=1}^{N} |P(f_i) − C_{k'}| + |C_{k'}|
        if Min ≤ r then
            T ← C
    return T
```

TABLE I: Flow Table for Fig. 2

| Flow | Source | Dest |
|---|---|---|
| $f_1$ | $s_1$ | $t_1$ |
| $f_2$ | $s_1$ | $t_2$ |
| $f_3$ | $s_1$ | $t_4$ |
| $f_4$ | $s_1$ | $t_5$ |

| Flow | Source | Dest |
|---|---|---|
| $f_5$ | $s_4$ | $t_2$ |
| $f_6$ | $s_5$ | $t_2$ |
| $f_7$ | $s_2$ | $t_2$ |
| $f_8$ | $s_3$ | $t_3$ |

Now, we have the features of each flow. According to k-means clustering, for $\kappa = 4$, we get clusters $\{1, 2, 3\}$, $\{1, 2, 6, 12\}$, $\{4, 6, 10\}$, and $\{14, 7, 3\}$.

**Theorem 1.** *The complexity of Algorithm 1 $O(K^2|E|I)$.*

*Proof.* The complexity of finding a path is $O(|V|+|E|)$ using BFS. Updating the capacity of each link and adding links to the bucket takes $O(|E|)$. The complexity of finding all the paths of $K$ flows is $O(K(|V|+|E|))$. Therefore, finding features of all flows takes $O(K(|V| + |E|))$. The most time consuming part of this algorithm is the k-means clustering which takes $O(K|E|I)$ , where $I$ is the number of iterations to converge. Therefore, the algorithm takes $O(N^2|E|I)$ □

## V. Tunnel ID Assignment

### A. Problem II Definition

***Problem II:*** *Find an ID assignment so that the number of forwarding rules is minimum.*

In this problem, we assume that the tunnels are already created and we need to assign IDs to them in such a way that the total number of rules is minimized. There are $|T|$ number of tunnels $T = \{t_1, t_2, t_3, ...\}$. We need to find a mapping

$\alpha : T \rightarrow Z^+$ so that the total number of rules for all tunnels ($|\rho(\alpha)|$) is the minimum. The problem can be expressed as follows:

$$\begin{aligned} &\text{minimize} \quad |\rho(\alpha)| \\ &\text{subject to} \quad \forall_{t \in T} \forall_{n_i \in t} \exists_{r \in \rho(\alpha)} \textsc{Output}(r) = n_{i+1} \end{aligned} \tag{3}$$

Here, $\rho(\alpha)$ denotes the set of rules for $\alpha$ ID mapping. Each tunnel $t \in T$ is a ordered set of nodes $n_1, n_2, ....$ $\textsc{Output}(r)$ denotes the action of rule $r$ which is next hop node to forward. The constrains implies that there must be a rule for each node of the tunnel. To find the best ID assignment function, we first start with a simple tunnels structure where all of the tunnels travel through a specific node. Then we use this basic solution to solve general cases.

### *B. Finding the Best $\alpha$ in Simple Structure*

In this subsection, we present an ID assignment method for simple structure. A simple structure is a group of tunnels where there exists a node that is a part of all of the tunnels in the group. We denote the node that is a part of all tunnels as the break-point. We divide the tunnel ID into two parts: ID prefix and ID suffix. All of the nodes before the break-point (pre-breakpoint nodes) use the ID prefix as matching criteria. This is because there is no need to output to multiple ports at pre-breakpoint nodes and all of the flows that are travelling through these tunnels are moving towards the break-point node. Therefore, the pre-breakpoint nodes will have forwarding rules in this format: "if *Tunnel ID* starts with *ID Prefix*, then output to port *X*". The value of the ID Prefix will be assigned in the complex structure.

All of the nodes after the break-point node including it-self (post-breakpoint nodes) may have multiple outputs and matching criteria based on the tunnels. We reserve some bits for each post-breakpoint nodes in the ID suffix. The number of reserved bits is the number of bits needed to represent the used output ports. For example, if the number of output ports is three, we need two bits to represent the output ports. If there is only one output port, we do not need to reserve any bits in the ID suffix. Therefore, the post-breakpoint nodes will have forwarding rules in this format: "if *Tunnel ID* starts with *ID Prefix* + *a prefix of ID Suffix*, then output to port *X*".

### *C. An Example of Simple Structure*

Let us consider the example in Fig 3(b). There are three tunnels, $t_1 : A \rightarrow S$, $t_2 : A \rightarrow Y$, and $t_3 : B \rightarrow X$. The break-point node is $D$. Therefore, the pre-breakpoint nodes are $A$ and $B$. The post-breakpoint nodes are $D$, $L$, $S$, $R$, $X$, and $Y$. Let the ID prefix of these tunnels is 0. Now, we calculate the number of reserve bits and ID suffixes for these three tunnels. At node $D$, there are two different output ports used. Therefore, we need 1 bit to represent the output ports. We represent the port of $D$ which is connected to $L$ and $R$ by 0 and 1, respectively. At node $L$, there is only one output port used and we do not need to reserve any bits in the ID suffix. At node $R$ there are two output ports used and the number of the reserved bits is one. Similarly, we represent the port of $R$ which is connected to $X$ and $Y$ by 0 and 1, respectively.

Tunnel $t_1$ travels through port 0 of node $D$. Here, we ignore the pre-breakpoint nodes, nodes using one output port, and the last node. This is because the pre-breakpoint node uses the ID prefix for matching. The nodes with one output port outputs all of the flows to the same port. The last node forwards the flows using regular rules. Therefore, the ID suffix for $t_1$ is 0. Tunnel $t_2$ travels through port 1 of node $D$ and port 1 of node $R$. Therefore, the ID suffix for $t_2$ is 11. Similarly, the the ID suffix for $t_3$ is 10. Therefore, the ID of the tunnels $t_1$, $t_2$, and $t_3$ are 00, 011, and 010, respectively. The matching criteria for $t_1$ at nodes $A$, $D$, and $L$ are starts-with 0, 00, and 00, respectively. The matching criteria for $t_2$ at nodes $A$, $D$, and $R$ are starts-with 0, 01, and 011, respectively. Similarly, the matching criteria for $t_3$ at nodes $B$, $D$, and $R$ are starts-with 0, 01, and 010, respectively.

### *D. Finding the Best $\alpha$ in Complex Structure*

In this subsection, we present an ID assignment method for complex tunnel structure. A complex structure is a group of tunnels where there exists no node that is a part of all of the tunnels in the group. We split the set of tunnels in such a way that each split becomes a simple structure. We convert the splitting problem to a classic set cover problem. Each node in the structure is considered as a set and the tunnels go through the nodes that are the elements of that set. We use the greedy solution to find a cover and the cover nodes are considered as the break-point nodes. We create a simple structure from the tunnels that go through each cover node. If any tunnels are already a part of the created simple structure, it is not considered twice.

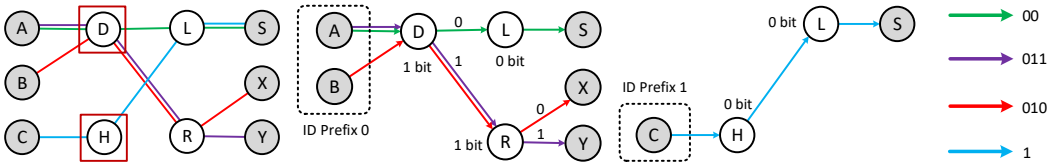
After getting the simple structures we assign ID prefixes to them. The number of bits needed to represent the number of structures is the length of the ID prefixes. For example, if there are four simple structures, the length of the ID prefixes is two bits. We assign unique ID prefixes to each simple structure. After that we apply the tunnel ID assignment method to the simple structure.

### *E. An Example of Complex Structure*

There are four tunnels, $t_1 : A \rightarrow S$, $t_2 : A \rightarrow Y$, $t_3 : B \rightarrow X$, and $t_4 : C \rightarrow X$ in Fig 3(a). There is no break-points in this structure so that we can call it s Complex Structure. We formulate a set cover problem where the universal set $U$ is $\{t_1, t_2, t_3, t_4\}$. Each node is a subset of $U$. For example, subset $A = \{t_1, t_2\}$ and subset $B = \{t_2\}$. We find the set cover to be $\{D, H\}$. Therefore, we need to create two simple structures and one bit ID prefix is required. The simple structure created from set $D$ is shown in Fig. 3(b) and we set 0 as ID prefix. The simple structure created from set $H$ is shown in Fig. 3(c) and we set 1 as the ID prefix. After that we assign the tunnel IDs using the method for the simple structure. Final ID assignment is shown in Fig. 3(d).

**Theorem 2.** *The complexity of Algorithm 3 is $O(|V|^2|T|)$.*

*Proof.* To find the complexity of Algorithm 3, we need to find the complexity of Algorithm 2. Let us assume that we




(a) All tunnels and cover nodes. (b) Forwarding IDs for $D$. (c) Forwarding IDs for $H$. (d) ID assignment.

Fig. 3: An example of tunnel ID assignment.

**Algorithm 2** Find tunnel ID for simple structure.

**Input:** Simple structure $G(V, E)$, tunnels $T$, ID prefix $P_{ID}$.
**Output:** An ID assignment $A : T \to R$.

1: **Procedure:** SIMPLE-ID-ASSIGNMENT$(G, T, P_{ID})$
2: $t \leftarrow$ any element in $T$.
3: **while** $n \in$ PATH$(t)$ **do**
4: **if** TUNNELS$(n) = T$ **then**
5: $B \leftarrow n$
6: **break**
7: ENQUEUE$(Q, B)$
8: **while** $Q \neq \emptyset$ **do**
9: $n \leftarrow$ DEQUEUE$(Q)$
10: **if** CHILD $(n) > 1$ **then**
11: $i \leftarrow 0$
12: **for** $c \in$ CHILD$(n)$ **do**
13: $S_{ID}[c] \leftarrow (i)$, $i \leftarrow i + 1$
14: **for** $t \in T$ **do**
15: $A(t) \leftarrow P_{ID}$
16: **for** $n \in$ PATH$(t)$ **do**
17: $A(t) \leftarrow A(t) + S_{ID}[n]$
18: **return** $A$

**Algorithm 3** Find tunnel ID for complex structure.

**Input:** Complex structure $G(V, E)$, tunnels $T$.
**Output:** An ID assignment $A : T \to N$.

1: **Procedure:** COMPLEX-ID-ASSIGNMENT$(G, T)$
2: $\forall_{n \in V} S_n \leftarrow$ TUNNELS $(n)$
3: $S \leftarrow \{S_n : n \in V\}$
4: $C \leftarrow$ set cover of $S$.
5: Create simple structure $G_c(V, E)$ from each $c \in C$.
6: $\forall_{c \in C} A_c \leftarrow$ SIMPLE-ID-ASSIGNMENT$(G_c(V, E))$
7: $A \leftarrow \bigcup_{c \in C} A_c$
8: **return** $A$

pre-compute the PATH(T) for all tunnels. This pre-compute takes $O(|T||V|)$ time. Finding a break-point node (Step 3-6) takes $O(|T||V|)$ time. Then, $O(|V|)$ time is needed to assign the ID-suffix to the post-breakpoint nodes. To find the ID of a tunnel, the algorithm iterates through the nodes on the path, which takes $O(|V|)$ time in the worst case. Therefore, for all tunnel it takes $O(|T||V|)$ times. In total the algorithm takes $O(|T||V|)$ times to find ID assignment of a simple structure.

In Algorithm 3, to find the cover set it takes $O(|V||T|)$ times. This is because there are $|V|$ subsets and each subset can hold at most $|T|$ elements. In the worst case, the number of elements in a cover set can be $|V|$. Therefore, for all of the cover sets take $O(|V|^2|T|)$ time to find the ID assignment. □

Next, we calculate the approximation ratio of Alg. 3. Alg. 3 uses Alg. 2 and the performance of Alg. 3 is a combination of both. We prove that Alg. 2 produces the optimal number of rules and calculate the approximation ratio of Alg. 3.

TABLE II: Topology Parameters

| Number of | Topology I (T-I) | Topology-II (T-II) |
|---|---|---|
| Nodes/ Sources /Destinations | 73/13/15 | 121 /16/24 |
| SDN switches | 45 | 81 |
| Links | 186 | 298 |

**Theorem 3.** *The approximation ratio of Alg. 3 is* $(ln|T|+1)$.

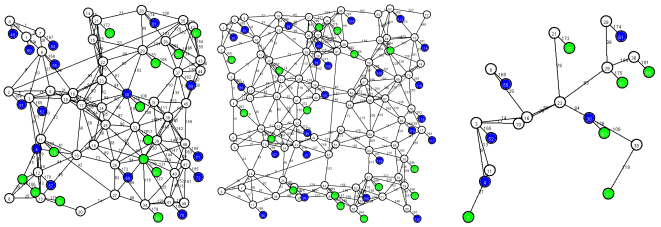
*Proof.* To calculate the approximation ratio of Alg. 3, we need to prove that Alg. 2 produces an optimal ID assignment which means that the number of rules needed to forward the flows of all tunnels is the minimum. The post-breakpoint nodes that have multiple outputs must have at least the same number of rules as the output ports. A single rule is enough for the nodes that have one output ports. On the other hand, a rule cannot selectively forward packets to multiple ports. According to our system model a rule is used to forward to each port with a prefix matching criteria. Therefore, the total number of rules is equal to the number of forwarding ports used. The approximation ratio of set cover greedy solution is $(ln|T|+1)$ [20]. Therefore, the approximation ratio of Algorithm 3 is $(ln|T| + 1)$, which is $O(ln|T|)$. □

## VI. SIMULATION

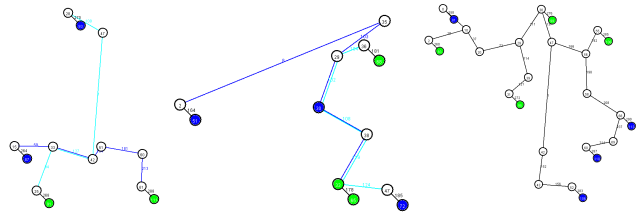
### A. Experimental settings

We conduct all of the simulations with our custom built java simulator. We want to count the number of rules, tunnels, groups for forwarding the flows. We do not need to analyze the real transmission time, actual link bandwidth, or packet drop scenarios. The network topologies we consider in this simulation contain hundreds of SDN switches, links, flows, sources, and destinations . Using NS3 or other similar simulators for this kind of large topologies would take a long time to produce results. This is the reason for building our own java simulator to get the results in short period.

We generate random topologies by taking an area of $500 \times 500$ square units. We divide the area into $50 \times 50$ blocks. In each block, a certain number of nodes are placed at random locations. We restrict nodes from being placed too close to other. Then, edges are generated based on distance. When the distance between two nodes is less than a threshold (70-90 units), we add an edge between them. After that few edges are added by picking up a pair of nodes randomly. After that, we attach a source or destination to some of the randomly selected nodes. We set the capacities of each as 100 Mbps. It should be noted that in a real-world network, the actual link capacity varies (100Mbps/1Gbps/10Gbps); we keep them the same for simplicity. If the link capacity is higher, then we need



(a) Topology T-I. (b) Topology T-II. (c) A complex structure in T-I.



(d) A complex structure in T-II. (e) A simple structure in T-I. (f) A complex structure in T-II.

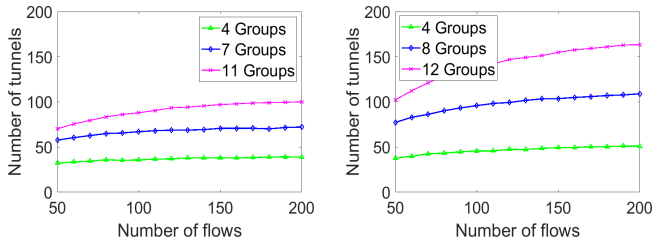
Fig. 4: Topologies, simple, and complex structure.

a higher number of flows or flows with higher rates to congest some of the links. This increases complexity and hardens the proper parameter selection (max data rate of flows, number of flows, congestion threshold.) Table II shows the properties of the topologies in details. Then, we generate a specific number of flows. We first generate all possible pairs of sources and destinations. After that, the specific number of pairs are chosen randomly and a flow is created. We set the minimum rate of a flow to 1 Mbps and the maximum rates are different for different simulations. The flows that do not travel trough any tunnel are routed using shortest path routing. Appropriate rules are added to the nodes for forwarding those flows.

Once the topology and flows are created, we divide the nodes into groups using the method described in Subsection IV-B. After that, tunnels are created between each pair of groups and a tunnel ID is assigned to each tunnel using Alg. 3. Then we count the number of rules after applying the tunnel rules. We measure the number of rules increased/flow, number of flows/tunnel for different numbers of flows, maximum data rate, and nodes in a network. We compare the number of rules/flow with the shortest path routing approach. All of the results in the plot are the average of 100 runs. Each type of simulation is executed with three different numbers of groups. The numbers of groups for different typologies are not same because of uncontrolled behavior of the number of groups in hierarchical clustering. We set a different cutting distance for the dendrogram of the hierarchical clustering and get different number of clusters. For topologies I and II, we have $\{3, 5, 6, 11\}$ and $\{3, 5, 9, 12\}$ groups, respectively.

### *B. Simulation result*

Firstly, we observe the effects on numbers of tunnels for different group settings and flows. Figs. 5(a) and 5(b) show the number of tunnels for different number of flows in T-I and T-II. We can observe that for both topologies, the number of tunnels increases with the increase in the number of flows.




(a) Different of flows in T-I. (b) Different of flows in T-II

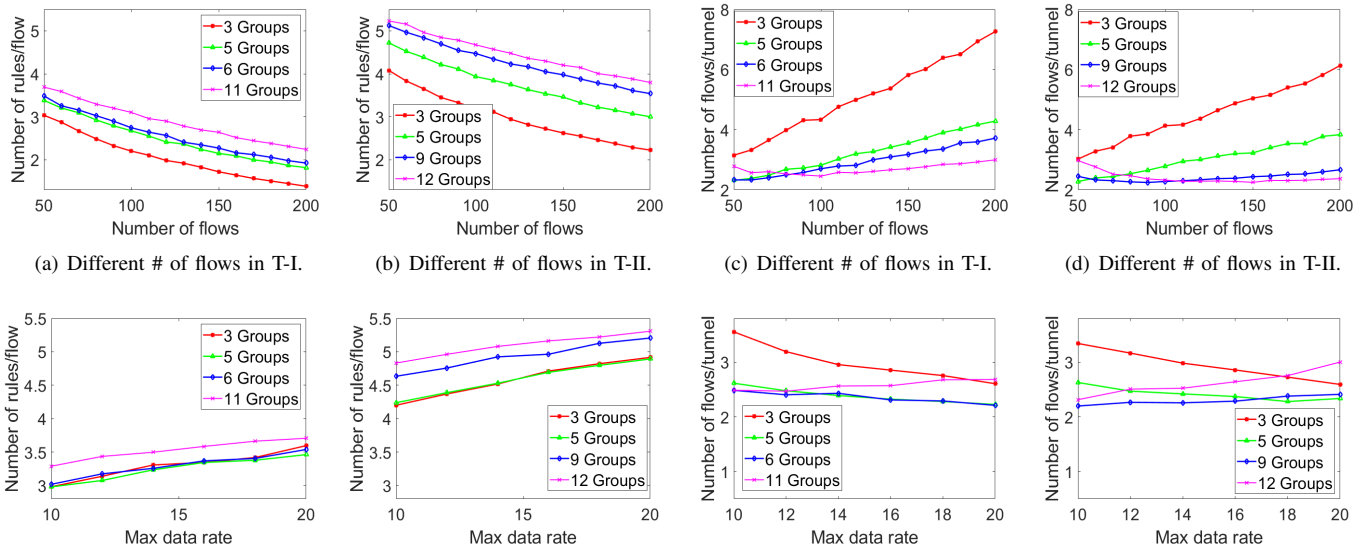
Fig. 5: Effects on number of tunnels.

After a certain number of flows, the increase of number of tunnels reduces. This is because more flows are using existing tunnels. When the number of groups is higher, the number of tunnels is also higher in both topologies.

We observe the number of rules/flow and number of flows/tunnels in our proposed approach by changing the number of flows in Topology I. Fig. 6(a) shows the number of rules/flow for different numbers of flows in T-I. We vary the numbers of flows from 50 to 200 and keep the maximum data rate at 10 Mbps. The number of rules/flow decreases with the increase of the number of flows. A higher number of groups produces a higher number of rules/flow. This is simply because when the number of flows are higher, the network needs a higher number of rules to forward them including a higher number of tunnels. If the number of groups is higher, then the number of flows between a couple of groups decreases and the number of flows per tunnel increases. As a result, the number of rules needed increases.

When the number of flows is 50, the average number of needed rules for a flow is 3.03 for 3 groups. When the number of groups is 11, the average number of needed rules for a flow is 3.69. The number of needed rules/flow increased by about 21%. When the number of flows is 200, the average number of rules needed to forward a flow is 1.38 for 3 groups. When the number of groups is 11, the average number of rules needed rules to forward a flow is 2.24. The number of rules/flow increased by about 62% and decreases about 54% and 39% for an increase of 150 flows for 3 and 11 groups, respectively.

Fig. 6(b) plots the number of rules/flow for different numbers of flows in T-II. We keep the same settings as previous for this simulation. We observe a similar behavior of the number of rules/flow as previous. The number of rules/flow is higher than that of in T-I. This is because T-II has higher number of nodes and links. As a result, the number of hops is higher and more rules needed to forward the flows. When the number of flows is 50, the average number of needed rules for a flow is 4.07 for 3 groups. When the number of groups is 12, the average number of needed rules for a flow is 5.22. The number of needed rules/flow increased by about 28%. When the number of flows is 200, the average number of rules needed to forward a flow is 2.22 for 3 groups. When the number of groups is 12, the average number of rules needed rules to forward a flow is 3.79. The number of rules/flow increased by about 70%. The number of rules decreases about 45% and 27%




(a) Different # of flows in T-I. (b) Different # of flows in T-II. (c) Different # of flows in T-I. (d) Different # of flows in T-II.

(e) Different max datarate in T-I. (f) Different max datarate in T-II. (g) Different max datarate in T-I. (h) Different max datarate in T-II.

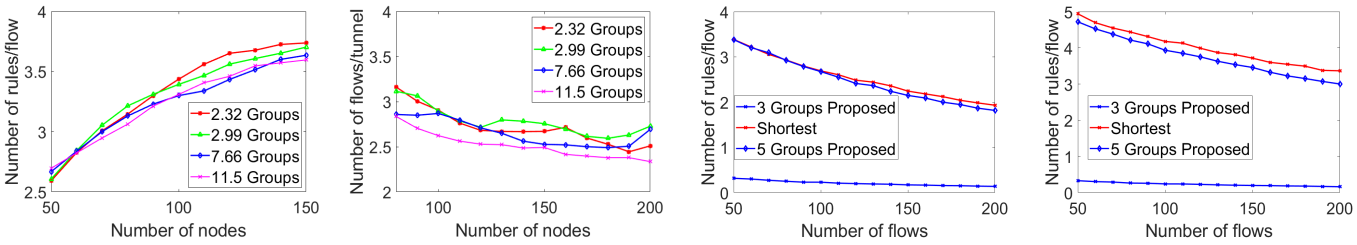
Fig. 6: Simulation results for T-I and T-II.

for an increase of 150 flows for 3 and 11 groups, respectively.

Figs. 6(c) and 6(d) shows the number of flows/tunnel for different numbers of flows in T-I and T-II, respectively. We keep the same settings as previous for this simulation. For all number of groups except 11 (in T-I) and 12 (T-II), the number of flows/tunnels increases with the increase of the number of flows. This is because the additional flows try to use the already used links. As we form tunnels with the used links, the number of flows in a tunnels increases. We can also observe that the number of flows/tunnels decreases when the number of groups increases. This is because when the number of groups are higher, the number of tunnels are also higher. As a result, the number of flows/tunnels decreases with the number of groups. We observe different behavior for groups 11 (in T-I) and 12 (in T-II). The number of flows/tunnel decreases upto a certain number of flows, then it continue increasing. When the number of flows is small and there is a higher number of groups, there exists some pairs of groups with no flows between them. Therefore, no tunnels are created. When the number of flow increases, the number of such pairs of groups decreases and the number of tunnels increases. When the number of such pair of groups becomes the minimum (zero), the the number of increased flows uses some of the existing tunnels and the flows/tunnel increases.

In T-I, when the number of flows is 50, the average number of flows/tunnel is 3.15 for 3 groups. When the number of groups is 6, the average number of flows/tunnel is 2.33. The number flows/tunnel decreased by about 26%. When the number of flows is 200, the average number of flows/tunnel is 7.26 for 3 groups. When the number of groups is 6, the average number flows/tunnel is 3.71. The number of flows/tunnel decreased by about 48%. The number flows/tunnel increases about 130% and 59% for an increase of 150 flows for 3 and 6 groups, respectively. In T-II, when the number of flows is 50, the average number of flows/tunnel is 3.02 for 3 groups. When the number of groups is 9, the average number of flows/tunnel is 2.45. The number flows/tunnel decreased by about 18%. When the number of flows is 200, the average number of flows/tunnel is 6.1 for 3 groups. When the number of groups is 9, the average number flows/tunnel is 2.66. The number of flows/tunnel decreased by about 56%. The number flows/tunnel increases about 101% and 8% for an increase of 150 flows for 3 and 9 groups, respectively.

Figs. 6(e) and 6(f) show the number of rules/flow for different maximum data rate of flows in T-I and T-II, respectively. We keep the number of flows at 150 and vary the maximum data rate from 10 to 20 Mbps. For all number of groups, the number of rules/flow increases with the increase of the number of flows. This is because when the data rate is higher, usage of some of the links become above threshold. As a result, some of the flows needs to use new links which increases the number of rules. The higher number of groups produce higher number of rules/flow for the same reason mentioned earlier. In T-I, when the maximum data rate is 10 Mbps, the average number of needed rules for a flow is 2.91 for 3 groups. When the number of groups is 11, the average number of needed rules for a flow is 3.28. The number of needed rules/flow increased by about 12%. When the maximum data rate is 20 Mbps, the average number of rules needed to forward a flow is 3.59 for 3 groups. When the number of groups is 11, the average number of rules needed to forward a flow is 3.70. The number of rules/flow increased by about 3%. The number of rules increases about 23% and 12% for an increase of 10 Mbps of maximum data rate for 3 and 11 groups, respectively. In T-II, when the maximum data rate is 10 Mbps, the average number of needed rules for a flow is 4.20 for 3 groups. When the number of groups is 12, the average number of needed rules for a flow is 4.8. The number of needed rules/flow increased by about 14%. When the maximum data rate is 20 Mbps, the average number of rules needed to forward a flow is 4.9 for 3




(a) Different # of nodes. (b) Different # of nodes. (c) Different # of flows in T-I. (d) Different # of flows in T-II.

Fig. 7: More simulation results for T-I and T-II.

groups. When the number of groups is 12, the average number of rules needed rules to forward a flow is 5.30. The number of rules/flow increased by about 8%. The number of rules increases about 16% and 10% for an increase of 10 Mbps of maximum data rate for 3 and 12 groups, respectively.

Figs. 6(g) and 6(h) shows the number of flows/tunnel for different maximum data rates in T-I and T-II, respectively. We keep the same settings as previous for this simulation. For all number of groups except 11 (in T-I) and 12 (T-II), the number of flows/tunnels decreases with the increase of the maximum data rate. this is because when the data rate is higher, a higher number of links are used and higher number of tunnels are created. The number of flows using tunnels also decreases because less flows use common paths. As a result, the number of flows/tunnel decreases with the increase of maximum bandwidth. For the number of groups 11 (in T-I) and 12 (T-II), we observe different behaviors. This is because the number of tunnels reduces when maximum data rate and number of groups are high. In this situation, the number of flows between a pair of groups is low and they use higher links than that of the low data rate flows. As a result, common paths and the number of tunnels decrease. Therefore, the number of flows/tunnel diereses. In T-I, when the maximum data rate is 10 Mbps, the average number of flows/tunnel is 3.55 for 3 groups. When the number of groups is 6, the average number of flows/tunnel is 2.4. The number flows/tunnel decreased by about 32%. When the number of flows is 20 Mbps, the average number of flows/tunnel is 2.60 for 3 groups. When the number of groups is 6, the average number flows/tunnel is 2.20. The number of flows/tunnel decreased by about 15%. The number flows/tunnel decreases about 26% and 8% for an increase of 10 Mbps of maximum data rate for 3 and 6 groups, respectively. In T-II, when the maximum data rate is 10 Mbps, the average number of flows/tunnel is 3.34 for 3 groups. When the number of groups is 9, the average number of flows/tunnel is 2.19. The number flows/tunnel decreased by about 34%. When the number of flows is 20 Mbps, the average number of flows/tunnel is 2.59 for 3 groups. When the number of groups is 9, the average number flows/tunnel is 2.41. The number of flows/tunnel decreased by about 6%. The number flows/tunnel decreases about 22% and 9% for an increase of 10 Mbps of maximum data rate for 3 and 9 groups, respectively. The number flows/tunnel increases about 8% and 30% for an increase of 10 Mbps of maximum data rate for 3 and 9 groups, respectively.

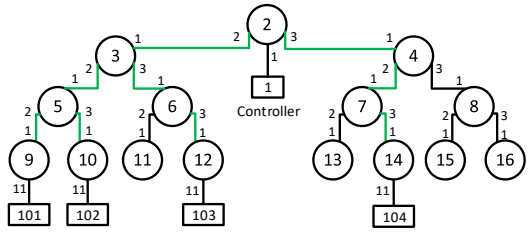
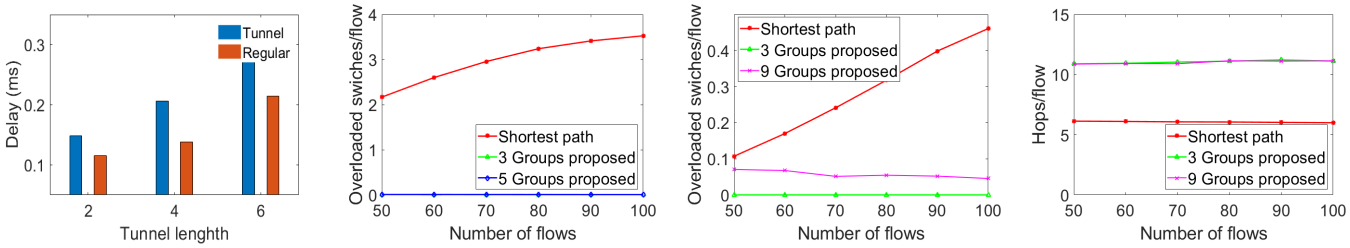



Fig. 8: Experimental topology.

Figs. 7(a) and 7(b) show the number of rules/flow and flows/tunnel for randomly generate topologies of different size. We keep the number of flows at 150 and the maximum data rate as 10 Mbps. We vary the number of nodes from 50 to 150 and observe the number of rules/flow and flows/tunnel. For all number of groups, the number of rules/flow increases with the increase of the number of nodes in topology. This is because when the network is larger, number of hops is also higher. As a result, the flows needs to use a higher number of rules. We can also observe that, in a smaller network with small number of groups has a lower number of rules/flow and higher flows/tunnel. Similarly, a large network with large number of groups has a lower number of rules/flow and higher flows/tunnel. When the topology is small (50 number of nodes), a smaller number of groups (average 2.32) produce 2.59 number of rules/flows where large number of groups (average 11.5) produces 3.84 rules/flows. When the topology is large (150 number of nodes), a smaller number of groups (average 2.32) produces 3.73 rules/flows where large number of groups (average 11.5) produce 3.59 rules/flows.

Figs. 7(c) and 7(d) compares the proposed approach with the shortest path routing for different numbers of flows in T-I, and T-II, respectively. We vary the number of flows from 50 to 200, while keeping the maximum data rate at 10 Mbps. For all of the numbers of groups, the number of rules/flow decreases with the increase of the number of flows. We can observe that the number of rules/flow is much lower for 3 groups in both T-I and T-II. The number of rules/flows is slightly lower for 5 groups in both topologies. In T-I, when the number of flows is 50, the average number of needed rules for a flow is 3.38 using shortest path routing, which is 11% higher than the proposed approach with 3 groups. When the number of flows is 200, the average number of needed rules for a flow is 1.93 using shortest path routing, which is 39% higher than the proposed approach with 3 groups. In T-II, when the number of flows is 50, the average number of needed rules for a flow




(a) Delay comparison. (b) Overloaded nodes for capacity 10. (c) Overloaded nodes for capacity 15. (d) Difference in hop count.

Fig. 9: Experiments and more simulation in T-II.

is 4.97 using shortest path routing, which is 21% higher than the proposed approach with 3 groups. When the number of flows is 200, the average number of needed rules for a flow is 3.66 using shortest path routing, which is 64% higher than the proposed approach with 3 groups.

Therefore, from the above simulation we can conclude that the proposed tunnel based approach works better than shortest path routing in terms of the number of rules. It is also observed that small number of groups in small topology and large number of groups in large topology perform well.

## VII. Experiments

### A. Experimental Settings

We test the feasibility of the tunnel based approach in our datacenter. We wanted to observe whether the delay caused by the tunneling approach is higher than the delay caused by a higher number of rules or not. The datacenter is composed of 15 SDN switches and 33 servers. The partial topology is shown in Fig 8. Nodes 2 to 16 are Pica 8 (P-3297) SDN switches and $101 - 104$ are the Dell Poweredge II servers that are connected at the leaf level switches. We use ONOS (2.2.2) as the controller and use the reactive forwarding plugin to forward the packets. The numbers at the ends of each link denotes the port numbers where it is plugged. We create three flows $\{101 \rightarrow 102, 101 \rightarrow 103, 101 \rightarrow 104\}$ and record the ping delays (round trip delay) without creating the tunnels. After that, we create three tunnels $t_2 : 9 \rightarrow 10$, $t_4 : 9 \rightarrow 12$, and $t_6 : 9 \rightarrow 14$. We use the built-in tunneling mechanism which is provided by the PICOS in pica8 switches and use the tag field in a packet header as the tunnel ID. We compare the delays before and after creating the tunnels.

### B. Experimental and More Simulation Result

Fig. 9(a) shows the comparison between the tunnel based approach and regular routing by round trip time (RTT). This delay includes two additional hops outside the tunnels. If the flow between 101 and 102 travels through a 2-length tunnel, then the RTT is 0.14ms. If it travels without a tunnel, then the RTT is 0.11. We observe an increase of 0.03ms in delay. If the flow between 101 and 104 travels through a 6-length tunnel, then the RTT is 0.29ms. If it travels without a tunnel, then the RTT is 0.21. We observe an increase of 0.08ms in delay. Similarly, we get an increase of 0.07ms delay for the 4-hop tunnel. As the time is RTT, for one direction the delays are 0.015 (through 3 switches), 0.035 (through 5 switches), and 0.04 (through 7 switches) for 2,4, and 6 lengths tunnels. Therefore, the average delay for a single hop is 0.0006. The increased RTT for a single switch for having high number of rules is 1.27ms. Therefore, for one direction it is 0.63ms delay, which is equivalent to the delay increased by a 105-length tunnel. Therefore, we can conclude that the tunnel based approach can decrease delay dramatically by using a fewer number of rules within SDN switches.

Figs. 9(b) and 9(c) shows the number of overloaded nodes in T-II if the capacities of a node is 10 and 15, respectively. We choose the capacity low (10 or 15) because the numbers of flows, sources, and destinations are lower than the real network. We vary the number of flows from 50 to 100 and observe the average number of overloaded nodes on the path of each flow. The number of overloaded nodes increases with the number of flows for the shortest path approach but it remains almost constant for the group based tunneling approach. If the capacity is 10 and the number of flows is 50, the shortest path approach shows 2.1 more overloaded switches per flows. The number of increased hop for the group based approach is 4.76. According to our experiment, the shortest path flows will encounter delays of a $2.1 \times 105 - 4.76 = 215$-length path. If the capacity is 15 and the number of flows is 50, the shortest path approach shows 0.1 and 0.3 more overloaded switches per flows for 3 and 9 groups, respectively. According to our experiment, the shortest path flows will encounter delays of a $0.1 \times 105 - 4.76 = 5.74$-length path for 3 groups. For 5 groups, each flows in group based approach will encounter $4.76 - 0.03 \times 105 = 1.61$- length delays. In this situation the group based approach performs worse than the shortest path approach. When the number of flows increases, the number of overloaded switches increases in shortest path approach and delay increases dramatically.

Therefore, from the experiments and simulations we can conclude that because of higher number of overloaded switches the increased delay is much more than the delay increased by additional number of hops.

## VIII. Conclusion

Traffic engineering is an important issue in a datacenter. Existing works on traffic engineering in software defined networking datacenters do not consider the virtual tunneling approach to minimize the number of rules in software defined networking switches. If the switches store a large number of rules, then the transmission delay increases significantly. When

the number of rules in the fast accessible memory of an SDN switch is higher, then the power consumption is also higher. We consider this an important factor in traffic engineering that can reduce the number of rules. To do so, we propose and evaluate a grouping and a virtual tunnel based approach. Our simulation supports that the proposed approaches can minimize the number of rules that the shortest path routing.

## REFERENCES


[1] "Pica8 P-3297," https://docs.pica8.com/display/picos2102cg/Pica8+P-3297+Switch, 2014.

[2] J. Wu, "Adaptive fault-tolerant routing in cube-based multicomputers using safety vectors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 4, 1998.

[3] Jie Wu and Ke Huang, "The balanced hypercube: a cube-based system for fault-tolerant applications," *IEEE Transactions on Computers*, vol. 46, no. 4, pp. 484–490, 1997.

[4] J. Wang and I. C. Paschalidis, "Statistical Traffic Anomaly Detection in Time-Varying Communication Networks," *Transactions on Control of Network Systems*, vol. 2, no. 2, Jun 2015.

[5] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, Oct 2001.

[6] K. Argyraki and D. R. Cheriton, "Loose Source Routing As a Mechanism for Traffic Policies," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, Aug 2004.

[7] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path Splicing," in *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4. ACM, Aug 2008.

[8] W. Xu and J. Rexford, "MIRO: Multi-path Interdomain Routing," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Sep 2006.

[9] M. Sun, K. Shao, and L. Wang, "Minimizing flow rules for rerouting multi-flows in multi-failure recovery over sdn," in *Proceedings of the 2019 8th International Conference on Software and Computer Applications*, ser. ICSCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 555–559.

[10] C. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-aware single link failure recovery in hybrid sdn networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1086–1094.

[11] M. Sun, K. Shao, and L. Wang, "Minimizing flow rules for rerouting multi-flows in multi-failure recovery over sdn," ser. ICSCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 555–559.

[12] I. S. Petrov, "Algorithm for reducing the number of forwarding rules created by sdn applications," *Modelirovanie i Analiz Informatsionnykh Sistem*, vol. 26, no. 1, pp. 122–133, 2019.

[13] S. Kotachi, T. Sato, R. Shinkuma, and E. Oki, "Multicast routing model to minimize number of flow entries in software-defined network," in *20th Asia-Pacific Network Operations and Management Symposium*, 2019.

[14] Z. Zhao, W. Yang, and B. Wu, "Flow aggregation through dynamic routing overlaps in software defined networks," *Computer Networks*, vol. 176, 2020.

[15] Z. Li and Y. Hu, "Pasr: An efficient flow forwarding scheme based on segment routing in software-defined networking," *IEEE Access*, vol. 8, 2020.

[16] J. Zhang, Z. Guo, M. Ye, and H. J. Chao, "Smartentry: Mitigating routing update overhead with reinforcement learning for traffic engineering." New York, NY, USA: Association for Computing Machinery, 2020, p. 1–7.

[17] C. Chuang, Y. Yu, A. Pang, and G. Chen, "Minimization of tcam usage for sdn scalability in wireless data centers," in *2016 IEEE Global Communications Conference*, 2016, pp. 1–7.

[18] M. Sun, K. Shao, and L. Wang, "Minimizing flow rules for rerouting multi-flows in multi-failure recovery over sdn," in *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. New York, NY, USA: Association for Computing Machinery, 2019.

[19] I. S. Petrov, "Algorithm for reducing the number of forwarding rules created by sdn applications," *Modelirovanie i Analiz Informatsionnykh Sistem*, vol. 26, no. 1, pp. 122–133, 2019.

[20] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.


## BIOGRAPHIES OF AUTHORS

**Rajorshi Biswas** is a PhD student of Computer and Information Sciences at Temple University, Philadelphia. He achieved his bachelor's degree from Bangladesh University of Engineering and Technology, Bangladesh. He is currently researching at the Center for Networked Computing (CNC) which is focused on network technology and its applications. His research areas include Wireless Networks, Wireless Sensor Networks, Wireless Security, Cryptography, Cognitive Radio Networks etc. He published his research in many conferences and journals including IEEE ICPADS 2018, IEEE GLOBECOM 2019, IEEE ICC 2019, IEEE Sarnoff 2019, and Resilience Week 2019.

**Jie Wu** Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. His current research interests include mobile computing and wireless networks, cloud computing, and network trust and security. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing and IEEE Transactions on Service Computing. He is/was general chair/co-chair for IEEE MASS'06, IEEE IPDPS'08, IEEE DOCSS'09, IEEE ICDCS'13, ACM MobiHoc'14, ICPP'16, IEEE CNS'16, and WiOps'21, as well as program chair/co-chair for IEEE MASS'04, IEEE INFOCOM'11, CCF CNCC'13, and ICCCN'20. Dr. Wu is a Fellow of the AAAS and a Fellow of the IEEE.