Minimizing the Number of Rules to Mitigate Link Congestion in SDN-based Datacenters

Rajorshi Biswas[1], Jie Wu[2], and Yang Chen[2]
Information Sciences and Technology, Penn State Berks, Reading, PA, USA[1]
Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA[2]

***Abstract*—Link congestion due to regular traffic and link flooding attacks (LFA) are two major problems in datacenters. Recent usage growth of software defined networking (SDN) in datacenters enables dynamic and convenient configuration management that makes it easy to reconfigure the network to mitigate the LFA. The reconfiguration that redirects some of the traffic can be done in two ways: the shortest alternative path and the minimum changes in rule path. The SDN switches have a limited capacity for the rules and the performance dramatically drops when the number of stored rules is higher. Besides, it takes some time to adopt the changes by the SDN switches which causes interruption in flow. In this paper, we aim at minimizing the number of rule changes while redirecting some of the traffic from the congested link. We formulate two problems to minimize the number of rule changes to redirect traffic. The first problem is the basic and it considers a congested link and a flow to direct. We provide a Dijkstra-based and a rule merging based solution to the problems. The second problem considers multiple flows and we propose flow grouping and rule merging based solutions. We conduct extensive simulations and experiments in our datacenter to support our model.**

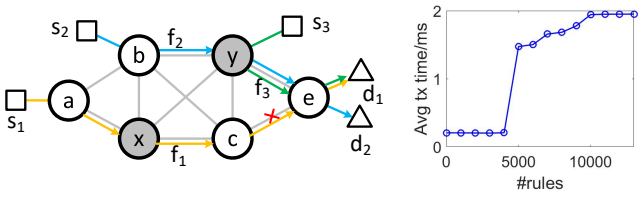
***Index Terms*—software defined networking, routing rules, link flooding attack, network security.**

I. Introduction

Congestion in links in a datacenter has become a common and serious problem nowadays. Link congestion can occur due to regular traffic or a link flooding attack (LFA). In LFA, an adversary aims to cut off an edge network from the Internet by generating traffic that is destined to decoy servers and the packets travels at least an important links on the paths to the victim. Nowadays, software defined networking (SDN) switches are taking place of the regular routers in datacenters. In the SDN architecture, a centralised software, called the controller, controls all of the actions of the SDN switches. This centralised architecture has opened opportunities to defend against the LFA and regular congestion easily and effectively.

Let us consider the network in Fig. 1(a) which is consist of four SDN switches (a, b, c, and e), two regular switches/routers (x and y), three sources (s_1, s_2, and s_3), and two destinations (d_1 and d_2). There are three flows $f_1 = s_1 \rightarrow d_1$, $f_2 = s_2 \rightarrow d_2$, and $f_3 = s_3 \rightarrow d_1$. Link (c, e) is congested by other flows as the result of extra traffic or link flood attacks which are not shown in the figure. We need to redirect flow f_1 to prevent link congestion. There are multiple possible redirection points and ways to redirect f_1. A redirection point is the farthest common SDN switch from the source between the new path

This research was supported in part by NSF grants CNS 2128378, CNS 2107014, CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, and CNS 1651947.



(a) Flow redirection. (b) Time vs. rules.

Fig. 1: (a) Flow redirection due to LFA and (b) transmission time vs. the number of rules.

(after redirection) and the old path. For f_1, the redirection points can be nodes a and c. If we consider node a as a redirection point, then we can redirect f_1 through the path $\{a, b, y, e, d_2\}$ and we need new rules at node a and b. The forward rule of f_1 from y to e already exists at y. Therefore, in this way, we need to add two rules and the number of increased hop is zero. If we consider node c as a redirection point, then we can redirect f_1 through the path $\{a, x, c, y, e, d_2\}$ and we need a new rule at node c. Therefore, in this way, we need to add a rule and the number of increased hops is one.

To observe the importance of the number of rules, we conduct a small experiment on a Pica8 P-3297 SDN switch at our datacenter. We observe that when the number of rules is greater than 4000 (see Fig. 1(b)), the transmission delay between two machines (2-hops away) jumps up about 1 ms. This happens when the fast accessible memory of an SDN switch becomes exhausted. This small experiment shows the importance of minimizing the number of rules and motivates us to conduct research on this topic.

In this paper, we focus on mitigating link congestion by redirecting some flows. To minimize the changes in rules, the flow may travel a longer path which may increase delay but overall delay due to no overloaded switches and interruption is low. We formulate a basic problem to illustrate the basic solution and the concept of core-tree (CT). This problem considers one congested link and one flow to redirect. We provide a Dijkstra-based solution for this problem. We formulate another problem for multiple flows redirection considering one congested link which is NP-hard and we provide a grouping-based solution. We provide extensive simulations and experimental results and compare it with existing approaches.

The remainder of this paper is arranged as follows. Section II presents some related works. In Section III, we present the system and the attacker model. In Sections IV and V, we present the problems of redirecting a flows and solutions. Section VI and Section VII present the simulation and experi-

mental and results. Finally, Section VIII concludes our paper.

II. Related Work

Unlike networks with regular topologies where re-routing under attacks and failures can be mitigated using inherit topology redundancy and structure [1], in regular networks with irregular topologies. There are three types of researches defending LFA and link congestion in SDN. Firstly, there are some statistical methods, including correlation, entropy, covariance, divergence, cross-correlation, and information gain-based system to detect attack traffic [2]. Other types of congestion mitigation systems include [3–6] that are based on multi-path routing and redirecting. These schemes either do not consider SDN switches or the number of rules.

Secondly, in [7], authors propose a hybrid and scalable SDN datacenter using both wireless and wired connections. In [8], authors propose an approach to ensure traffic reachability if any single link fails. They redirect traffic on the failed link to SDN switches via pre-configured IP tunnels. Some other works, such as [9], considers the problem of minimizing the number of rules in SDN switches and propose a heuristic algorithm that creates a reduced representation of rules in the SDN switches in network. In [10], authors propose a DoS attack on controller and link by dynamically re-routing potential malicious traffic, adjusting flow timeouts, and aggregating flow rules. In [11], the authors proposed an efficient flow forwarding scheme and an intelligent encoding algorithm to minimize the number of rules.

Finally, in [12], the authors considers multiple failures of links and multi-flows re-routing in SDN environment. They propose a model for communication overhead between controller and switch during flow re-routing to minimize flow rules. In [13], authors propose a multicast routing model for multiple multicast requests to minimizes the number rules. In [14], authors propose an ILP to minimize the total cost. The solutions of the problems considered in these works are based on integer linear programming.

We discussed three types of existing defense systems: (1) attack packet/flow detection at a router followed by packet drop, rate control, and redirection, (2) re-routing and avoiding the congested link using statistical and heuristic methods, and (3) using redirecting some flows using linear programming. The first type increases the router computation overhead and the false positive creates great loss to the regular users. Besides, bots are smart to create traffic similar to users and are almost unidentifiable using statistical techniques. For the second type, the heuristically methods performs poorly in special cases. The linear programming approach is time consuming if the topology is complex and big. Therefore, a traffic routing where the number of new rules is minimized with a low time complexity is necessary.

III. System Model

A. Network Model

Our network is composed of regular routers, SDN switches, sources, and destinations. We assume that the controller knows the congested/attacked links and the routing policy of both regular routers and SDN switches. A rule in an SDN switch is simply a packet forwarding policy. The properties of an incoming packet are matched with the criteria (source address, destination address, incoming port) of the rules and actions are taken according to the matched rule. The controller can add, modify, or delete rules from an SDN switch but not from a regular switch. If one or multiple links are congested because of regular traffic or attack traffic, some of the flows going through those links must be redirected through any non-congested path. After redirection, none of the links should be congested. To redirect a flow, the controller needs to add or modify rules in some of the SDN switches. The number of rules in an SDN switch is important because of the limited storage. When the number of rules is above a certain threshold, the packet forwarding delay increases dramatically. It also takes some time to add a rule in an SDN switch. If the number of additions or modifications is large, then the flow will be interrupted. The order of rule addition to the controller does not affect the interruption, because the rule are loaded to the switch only when there is a packet for forwarding at the switch. Therefore, we aim at minimizing the number of rules needed to redirect some flows from congested links.

Let a flow $f = s \rightarrow d \in F$ from source s to destination d travels through path $P = \{n_1, n_2, ...\}$. Here, n_i is the ith node on path and P is an ordered set. Each of the nodes in P has a rule for forwarding the packets in flow f. Let $R^f = \{r_1, r_2, ...\}$ be the set of rules needed to forward the packets of f. $R^f \in R$ where R is the set of rules for all flows in the topology. We will use the superscript f when it is necessary, otherwise we will omit the superscript. Rule r_1 resides in SDN switch n_1 and forwards the packets of f to node n_2 through the link (n_1, n_2). Some of the nodes in P are SDN switches and some of them are regular routers. We denote $SDN(n_1) = 1$ if n_1 is an SDN switch and $SDN(n_1) = 0$ if it is regular router. After redirection, the flow f travels through a new path $P' = \{n'_1, n'_2, ...\}$. The new set of rules is $R'^f = \{r'_1, r'_2, ...\}$ is the set of rules needed to forward the packets in f though P'. Therefore, the newly added rules are $R' - R$ and they must reside in SDN switches. This is because the controller cannot add any rules in a regular router. The number of newly added rules is $|R' - R|$ that we aim to minimize.

Therefore, our system is a four phase system. In the first phase, the controller detects the congested links. In the second phase, it selects the flows and new paths to redirect. In the third phase, it adds/modifies the forwarding rules to the SDN switches. At the final phase, it removes the flows that are not necessary from the SDN switches.

IV. Redirecting a flow

Problem I: *Find the route to redirect the flow so that the number of rules needed is the minimum.*

In this problem, we assume that, the controller knows the flow to redirect. Let $P = \{n_1, n_2, ...\}$ be the path of the flow $f = s \rightarrow d$. Let the link (n_c, n_{c+1}) on the path P be congested. After redirection, the new path is $P' = \{n'_1, n'_2, ...\}$.

R' denotes the set of rules needed to forward packets of the flow f through P'. In this case, the problem can be expressed as the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & |R' - R| \\ \text{subject to} \quad & |P'| - |P| \leq \delta_0 \\ & \forall_{1 \leq i \leq |P'|} \, (n'_i, n'_{i+1}) \neq (n_c, n_{c+1}) \\ & \forall_{n_i, n_{i+1} \in P'} \, r((n_i, n_{i+1})) \leq \delta_1 \end{aligned} \tag{1}$$

Here, R denotes the set of existing rules in the network. The first constraint means that the new path can be at most δ_0 longer than the old path. The second constraint ensures that the congested link does not appear on the new path. δ_1 denotes the maximum allowable bandwidth through the congested link and $r((n_i, n_{i+1}))$ denotes the data rate usage of link (n_i, n_{i+1}). The third constraint ensures that none of the links traveled by redirected flows are congested after redirection.

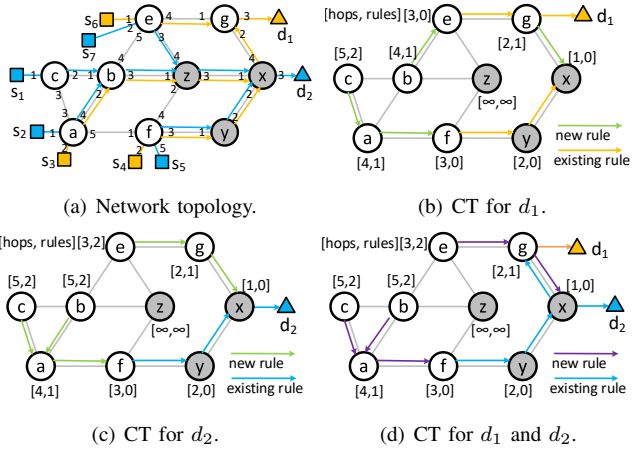
A. A Dijkstra-based Solution

The problem is the basic part of problem II (discussed in Section V) and easy to solve. This kind of situation may appears when a heavy flow congest a link. We illustrate the solution as a basic of problem II.

To solve the problem, the controller needs to formulate a CT for the destination of the flow. A CT is a minimum spanning tree covering a destination and the sources where distance is the number of new rules. Each node in the CT has the information of the next hop and the number of rules to needed to forward the flow through that node. The flow can be redirected from any upstream SDN switches that remain on the flow path. If the flow is redirected to any node of the CT, it is guaranteed to be delivered to the destination. To formulate a CT, the controller collects all of the rules which are responsible for forwarding the packets of the flow we are redirecting. The complete algorithm for creating a CT is shown in Alg. 1. The complexity of Alg. 1 is same as the Dijkstra algorithm, which is $O(|V|+|E|log|V|)$ ($|V|$ and $|E|$ are the number of vertices and edges, respectively).

B. An Example

Let us consider the example in Fig. 2(a). Let us assume that link (z, x) is congested and the flow $s_1 \rightarrow d_2$ needs to redirect. Flow $s_1 \rightarrow d_2$ can be redirected from either node b or a. To find the best location of redirection we need to formulate the CT for destination node d_2. Fig. 2(c) shows the CT for destination d_2. We first remove the congested link. Then, we calculate the number of hops and number of rules needed to deliver any flows to d_2. The number of rules gets a higher priority over number of hops. We use the Dijkstra algorithm to calculate the smallest rule path from each node. We do not consider the links that becomes congested if they pass $s_1 \rightarrow d_2$ flow. For regular nodes, we do not update distance if the next hop is different from the previous next hop. Construction of CT also considers the data rate of the flow. For example, if we are constructing a CT for redirecting a flow of 10 MBps, then links that can forward additional 10 Mbps traffic without being congested are considered. From the CT, we can see that if we redirect from node b, it will need 2 new rules. If we redirect from node a, it will need 1 new rule. Therefore, we add a new rule at node a that indicates if the source and destination are s_2 and d_2, respectively, then forward the traffic to node f.



(a) Network topology. (b) CT for d_1.

(c) CT for d_2. (d) CT for d_1 and d_2.

Fig. 2: An example for Problems I and II.

V. Redirecting Multiple Flows

Problem II: *Find a set of flows to redirect from all of the flows through a congested link so that the number of rules needed to redirect is the minimum.*

In this problem, we assume that the controller knows the congested link. Let F be the set of flows passing through the congested link (n_c, n_{c+1}). We split F into F_0 and F_1. Flows in F_0 will continue using the congested link. Flows in F_1 will be redirected. Let P'^f be the new path of the flow f ($f \in F$). In this case, the problem can be expressed as the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & |\cup_{f \in F_1} R'^f - R| \\ \text{subject to} \quad & \forall_{f \in F_1} |P'^f| - |P^f| \leq \delta_0 \\ & \forall_{1 \geq i \geq n'} \, (n'^f_i, n'^f_{i+1}) \neq (n_c, n_{c+1}) \\ & \forall_{f \in F_0} \, \forall_{n_i, n_{i+1} \in P'^f} \, r((n_i, n_{i+1})) \leq \delta_1 \end{aligned} \tag{2}$$

Here, $\cup_{f \in F_1} R'^f$ is the set of all rules needed to redirect the flows. If a rule is used to redirect multiple flows, then it appears on multiple rule sets. Therefore, we need to take the union of the set of rules of all flows. δ_1 denotes the maximum allowable bandwidth through the congested link. The first constraint means the new path can be at most δ_0 longer than the old paths. The second constraint ensures that the congested link does not appear on the new path. The third constraint ensures that none of the links traveled by redirected flows are congested after redirection.

A. A Flow Grouping Solution

The problem is NP-hard and we provide a heuristic solution for this problem. The basic idea is to group the flows and add some rules that can work on the group of flows. The small number of groups helps to reduce the run-time with some additional rules. We group the flows based on common upstream links. We consider the paths of the flows up-to k hops from the congested link. Flows having the same path goes to the same group. Therefore, each group has the same

Algorithm 1 Generate CT

Input: $G(V, E, R)$, congested links L_c, destination d.
Output: A CT of G for destination d.

Procedure: FIND-CT(G, L_c, d)
  $\forall_{n \in V} C[n] = \infty$, $D[n] = \infty$, $S[n] = \emptyset$
  $Q \leftarrow \{d\}$, $C[d] \leftarrow 0$, $E \leftarrow E - L_c$
  while $Q \neq \emptyset$ **do**
    $n \leftarrow$ vertex in Q with min $C[n]$
    REMOVE(Q, n)
    for $n' \in$ NEIGHBOR(n) **do**
      $r \leftarrow (d, n)$
      if $r \in R$ **then**
        $C[n'] \leftarrow$ MIN($C[n'], C[n]$) , $S[n'] \leftarrow S[n]$
        $D[n] \leftarrow D[$ARGMIN($C[n], C[n']$)$] + 1$
      else if n' is SDN **then**
        $C[n'] \leftarrow$ MIN($C[n'], C[n] + 1$), $S[n'] \leftarrow S[n] \cup \{r\}$
        $D[n] \leftarrow D[$ARGMIN($C[n], C[n']$)$] + 1$
  return $G_{ct}(V, E, C, S, D)$

Algorithm 2 Combine CTs

Input: $G(V, E, R)$, congested links L_c, destinations set D.
Output: A CT of G for destination d.

Procedure: MULT-DEST-CT(G, L_c, D)
  $\forall_{d \in D}$ $CT[d] \leftarrow$ FIND-CT(G, L_c, d)
  $\forall_{n \in V} C[n] = \infty$, $D[n] = \infty$, $S[n] = \emptyset$
  $Q \leftarrow D$, $E \leftarrow E - L_c$
  while $Q \neq \emptyset$ **do**
    $n \leftarrow$ vertex in Q with min $C[n]$
    REMOVE(Q, n)
    $X \leftarrow \bigcup_{d \in D} CT[d].S$, $C[n] \leftarrow |X|$
    for $n' \in$ NEIGHBOR(n) **do**
      $r \leftarrow (D, n')$
      $S[n] \leftarrow d \in DCT[d].S[n]$
      if $r \in \bigcup_{d \in D} CT[d].S[n]$ **then**
        $C[n] \leftarrow$ MIN($C[n], C[n']$)
        $D[n] \leftarrow D[$ARGMIN($C[n], C[n']$)$] + 1$
      else
        $C[n] \leftarrow$ MIN($C[n], C[n'] + 1$), $S[n] \leftarrow S[n] \cup \{r\}$
        $D[n] \leftarrow D[$ARGMIN($C[n], C[n']$)$] + 1$
  return $G_{ct}(V, E, C, S, D)$

upstream path up-to length k. Each node on the common path is eligible to be a redirect point. After that, a CT for the destinations in each group is created. Creating a CT for a group is basically combining the CTs for the destinations in that group. To combine multiple CTs, we need compute the number of rules needed and the hop-counts for the destinations. A node can forward the packets that belong to the destination group in two different ways:

1) **Separate forwarding:** The node forwards the packets of each flows to separate next hops that offers the least number of rules. The combined number of rules needed is the total of the number of unique rules of all CTs.
2) **Aggregated forwarding:** The node forwards all of the packets that are destined to any of the destinations in the group to a next hop. The combined number of rules needed is one plus the minimum of separate or aggregated forwarding of all CTs of the next hop nodes.

However, we select the option that provides the minimum number of rules. The number of CTs are the same as the number of groups. For each group, we calculate the minimum number of rules needed to redirect the flows from one of the redirect points. Then, the group with the minimum number of rules is taken out for redirection. After that, we update the CT of the rest of the groups. We need to update the CTs because after redirection, the capacity of some links will change. As a result, the CT for the rest of the groups may change. This process continues until the congested link becomes non-congested. Alg. 2 shows the complete procedure of combining multiple CTs.

TABLE I: Flows and bandwidth

	Flow	Bandwidth
f_0	$s_1 \rightarrow d_2$	40 Mbps
f_1	$s_2 \rightarrow d_2$	10 Mbps
f_2	$s_3 \rightarrow d_1$	20 Mbps
f_3	$s_6 \rightarrow d_1$	10 Mbps
f_4	$s_4 \rightarrow d_1$	10 Mbps
f_5	$s_5 \rightarrow d_2$	10 Mbps
f_6	$s_7 \rightarrow d_2$	20 Mbps

The value of k plays an important role in grouping. A small k produces a small number of large size groups. This reduces the running time of the algorithm, but a large group of flows may need more rules to redirect.

B. An Example of Common k Links Grouping

Let us consider the example in Fig. 2(a). Each link has a maximum capacity of 100 Mbps. Flows and their bandwidths are shown in Table I. We can observe that flows f_1, f_2, f_3, and f_6 are passing through link (z, x). Their total bandwidth is $40 + 10 + 20 + 20 = 90$ Mbps. We assume that any link utilization above 70% (70 Mbps) is considered congested. Therefore, only link (z, x) is congested in our example. We need to redirect some of the links among f_0, f_1, f_2, and f_6 to make link (z, x) usage less than or equal 70 Mbps.

Next, we group the flows according to the Common k Links policy. Let us consider $k = 1$ first. The the upstream path of the flows of length 1 is following:

$$f_0 : \{b, z\}, \quad f_1 : \{b, z\}, \quad f_2 : \{b, z\}, \quad f_6 : \{e, z\}.$$

We can observe that f_0, f_1, and f_2 have the same upstream path $\{b, z\}$. Therefore, flows f_0, f_1, and f_2 goes to group 1 and f_6 goes to group 2.

Next, we construct the CT for the groups $(k = 1)$. The CT for group 1 contains routes for destinations d_1 and d_2. The total bandwidth of the group is $40 + 10 + 20 = 70$. Therefore, any link having existing flows cannot redirect all of the traffic. As a result, group 1 cannot be redirected. The CT for group 2 contains routes for destination d_2 only. Therefore, the CT is the same as the one in Fig. 2(c). The redirect points of group 1 is e. The number of rules needed to redirect from e is 2. Therefore, the minimum number of rules to redirect the flows in group 1 is 2. At this point, we redirect the flows in group 2 and the number of rules needed is 2. After redirection, the congested link usage becomes $90 - 20 = 70$ Mbps and is no longer considered congested.

If we consider $k = 2$, the upstream paths of the flows of length 2 are the following:

$$f_0 : \{c, b, z\}, \quad f_1 : \{a, b, z\}, \quad f_2 : \{a, b, z\}, \quad f_6 : \{e, z\}.$$

Now we have a different grouping for $k = 2$ than $k = 1$. Therefore, flows f_0 goes to group 1, f_1, and f_2 go to group 2, and f_6 goes to group 3. The CT for group 1 contains routes for destination d_2 only. Therefore, the CT is the same as the one in Fig. 2(c). The redirect points of group 1 are c and b. The number of rules needed to redirect from b and c is 2.

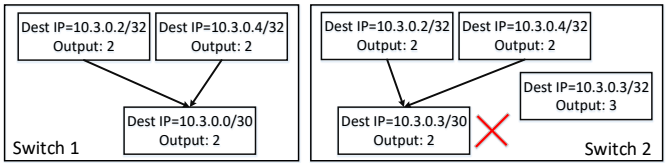


Fig. 3: Merging rules.

The minimum number of rules to redirect the flows in group 1 is 2. The CT for group 2 contains routes for destination d and d_2. Therefore, the CT is the same as in Fig. 2(d). The redirect points of group 2 are a and b. The number of rules needed to redirect from a and b is 1 and 2, respectively. So, the minimum number of rules to redirect the flows in group 2 is 1. The CT for group 3 contains routes for destination d_2 only. Therefore, the CT is the same as in Fig. 2(c). The redirect point of group 3 is e only. This is because, z is a regular router and cannot be redirected from there. The number of rules needed to redirect from e is 2. Therefore, the minimum number of rules to redirect the flows in group 3 is 2.

Therefore, redirecting group 2 adds the least number of rules. We redirect the flows f_1 and f_2 from node a. After redirection, the link capacity of the congested link comes down to $90-(10+20) = 60$ Mbps, which is below 70% of the link utilization and the algorithm stops.

Theorem 1. *The complexity of Alg. 2 is* $O(|D|(|V| + |E|log|V|))$.

Proof. To calculate the complexity of Alg. 2, we need to calculate the complexity of Alg. 1. Alg. 1 is basically the Dijkstra algorithm. Therefore, the complexity is $O(n+mlogn)$ where $n = |V|$ and $m = |E|$. Step 2 in Alg. 2 takes $O(|D| \times (|V| + |E|log|V|))$. Step 11 takes $O(|D|)$ times because at any node the number of added rules can be at most as the number of destination nodes. Therefore, the while loop in Step 5 takes $O(n \times |D|)$ time. So, the total run time of the Alg. 2 is $O(|D| \times (|V| + |E|log|V|)) + O(|V| \times |D|)$ which is $O(|D|(|V| + |E|log|V|))$. □

C. Greedy Rule Merging Solution

In this subsection, we consider that two rules can be merged if it does not conflict with others. We define conflict of rules for merging if all of the following conditions are satisfied:

1) Forwarding ports of the rules to be merged must be same.
2) Merged rule matching criteria cannot overlap with the criteria of any other rules except the rules to be merged.
3) Merged rule matching criteria must match all of the criteria of rules to be merged.

Let us consider the example in Fig. 3 to explain the merging of rules. Switch 1 has two rules that have the same output ports so they may be merged (condition 1 satisfied). We combine the matching criteria to 10.3.0.0/30 that covers both rules (condition 2 satisfied). As there are no other rules, condition 3 is satisfied. Therefore, we can merge the rules to *if destination IP is 10.3.0.0/30 then output to port 2*. In switch 2, we have three rules. Rule 1 and Rule 2 have same output ports. We combine their matching criteria to 10.3.0.0/30, which covers the matching criteria of rule 3 (violation of condition 3). Therefore, the rules cannot be merged.

When we add new rules and the new rules can be merged with any of the existing rules, then the number of rules does not increase. Using this principle, we design a greedy algorithm to solve problem II. The greedy algorithm is simple; we calculate the minimum number of rules for each flow through alternative possible paths with merging. Then, we pick the flow with the minimum number of unseen rules and redirect it through the path. After that, we mark the taken unseen rules as seen. We continue until the congested link becomes non-congested.

TABLE II: Topology Parameters

Number of	Topology I (T_1)	Topology-II (T_2)
Nodes	73	121
Sources/Destinations	13/15	16 /24
SDN switches	45	81
Links	184	296

D. An example

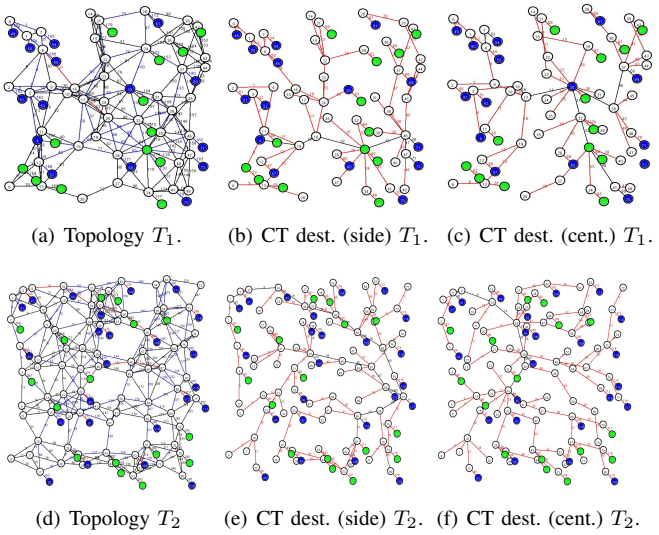
Let us consider the example in Fig. 2(a) again. We first consider f_0 through path $\{s_1, c, a, f, y, x, d_2\}$. There are already rules to forward packets to d_2 at node f, y, and x. We need to add two new rules at a and c. None of the existing rules output to port 3 at c and port 5 at a. Therefore, we need 2 rules to redirect f_0 through the path. If we consider path $\{s_1, c, b, e, g, x, d_2\}$, then we need new rules at nodes b, e, and g to output the packets to ports 4, 4, and 2 ,respectively. There are no rules at b and g to output to port 4 and 2. The existing rule at e that outputs to port 4 can be merged with the new rules. Therefore, we also need 2 new rules to forward f_0 through this path. Similarly, considering other paths, we find that the cost of redirecting f_0 is 1 ($\{s_1, c, b, z, f, y, x, d_2\}$). Similarly, we calculate the cost of redirecting f_1, f_2, and f_6 is 1. Therefore, we can pick any of the flows to redirect. Let us pick f_0 to redirect. After redirection we calculate the cost of the remaining flows and continue the process until link (z, x) becomes non-congested.

Theorem 2. *The complexity of Greedy Rule Merging Solution is* $O(|F||R_n|(|V| + |E|))$.

Proof. Finding the of cost of redirection needs to traverse the topology graph once. To find whether a new rule can be merged or not takes $O(|R_n|)$, where R_n is the set of existing rules at node n. Therefore, each iteration of the greedy algorithm takes $O(|R_n|(|V| + |E|))$. The iteration can run at most $|F|$ times. Therefore, the complexity of the algorithm is $O(|F||R_n|(|V| + |E|))$. □

Theorem 3. *The complexity of Greedy Rule Merging Solution has an approximation ratio of* $\eta(1 - \frac{1}{e})$.

Proof. To find the upper bound of the greedy approximation, we assume that all rules that output to the same port can be merged. We denote a rule by r_{np} where n is the node and p is the output port. We denote the minimum set of rules needed to redirect a flow f by S_f. Now, the problem



(a) Topology T_1. (b) CT dest. (side) T_1. (c) CT dest. (cent.) T_1.

(d) Topology T_2 (e) CT dest. (side) T_2. (f) CT dest. (cent.) T_2.

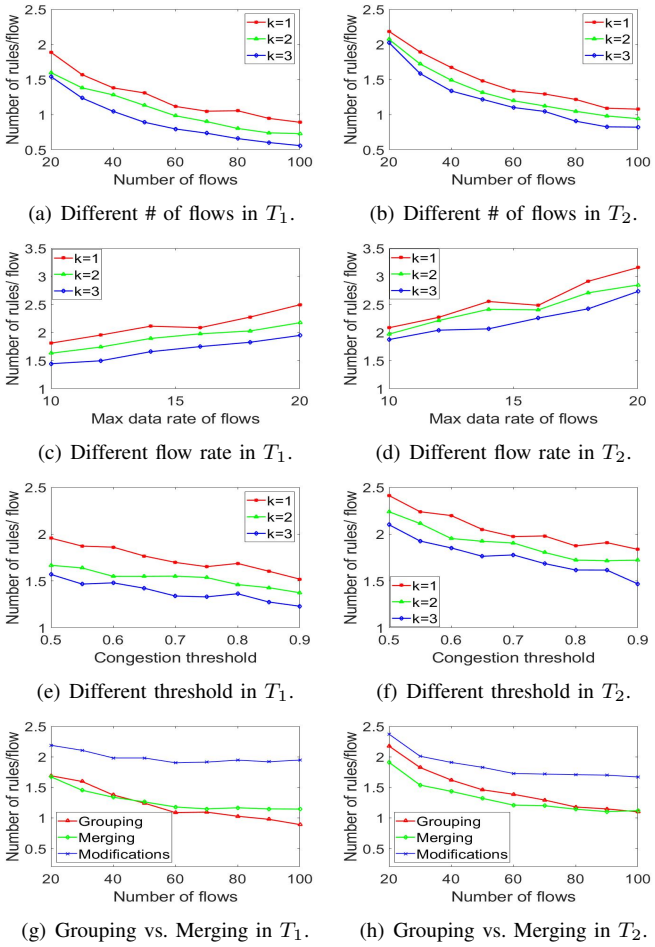
Fig. 4: CT of a randomly generated topology for different destinations (green node: source, blue node: destination, red link: new rules added, black link: existing rule).

II can be expressed as finding k elements from the set $S = \{S_1, S_2, ...S_{|F|}\}$ so that the number of unique elements is the minimum. Now we convert the problem to a maximum coverage problem by replacing S with S' and S_f with S'_f. Here $S' = \{S'_1, S'_2, ...S'_{|F|}\}$ and $S'_f = U - S_f$. U is the universal set ($U = \cup_{f=1}^{|F|} S_f$). Therefore, a maximum coverage solution represents a solution to Problem II. The maximum coverage greedy solution has an approximation ratio of $(1-\frac{1}{e})$. Let η be the merging ratio of the rules. If there are $|R_n|$ number of rules before merging and $|R'_n|$ number of rules that output to an specific port at node n, then $\frac{|R_n|}{\eta} = |R'_n|$. Therefore, the number of rules for the greedy rule merging solution must be less than $\eta(1 - \frac{1}{e})$ times of the minimum number of rules. □

VI. Simulations and Experiments

A. Simulation Settings

We conduct the experiments with our custom built java simulator. We want to count the number of rules needed for redirecting the flows. We do not need to analyze transmission time, actual link bandwidth, or packet drop issues. The network topologies we consider in this simulation contain $70 - 120$ SDN switches, sources, and destinations. Using NS3 or other similar simulators for this kind of simulation would take a long time to produce results. That is the reason for building our own java simulator, to get the results fast. We generate random topologies and set the link capacities as 100 Mbps. Although in a real network the link capacity is different for different links (100Mbps/1Gbps/10Gbps), we keep them the same for simplicity and ease of comparison. Fig. 4 and Table II show the structures and details properties of the topologies. Next, we generate the desired number of flows by randomly selecting a source and a destination with a random rate from a range. We set the minimum rate to 1 Mbps. We select different maximum rates for different simulations.



(a) Different # of flows in T_1. (b) Different # of flows in T_2.

(c) Different flow rate in T_1. (d) Different flow rate in T_2.

(e) Different threshold in T_1. (f) Different threshold in T_2.

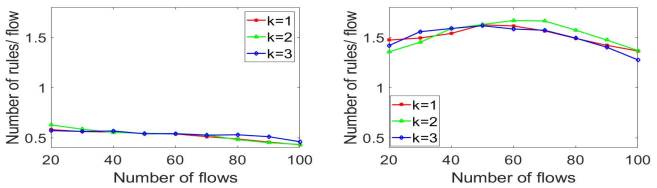
(g) Grouping vs. Merging in T_1. (h) Grouping vs. Merging in T_2.

Fig. 5: Number of rules per redirected flow.

We measure the number of rules increased/flow and the number of hops increased after redirection for different number of flows, maximum data rate, and congestion threshold. The congestion threshold is defined by $\frac{\delta_1}{\text{link capacity}}$. We compare the performance of our approaches with shortest alternative path redirection approach. All of the results in the plot are the average of 200 runs with three different k values.

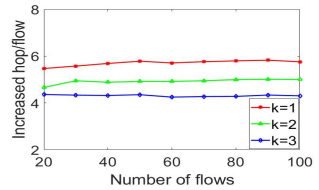
B. Simulation result

Firstly, we observe the number of increased rules per flows by changing different parameters. Figs. 5(a) and 5(b) show the number of increased rules per flow for different number of flows in T_1. We vary the number of flows from 20 to 100 and keep the maximum data rate as 10 Mbps. For all values of k, the number of rules per flow decreases with the increase of number of flows. A higher value of k, produces lower number of rules per flow. When the number of flows are higher, the network needs a higher number of rules to forward them. As a result, the existing rules help to forward the packet through an alternative way and we need fewer number of new rules. If k is higher, then the number of groups increases and the selection of flows to redirect is better. When the number of flows is 20 and $k = 1$ (or $k = 3$), the average number of needed rules to redirect a flow is 1.88 (or 1.54). The number

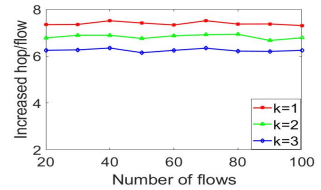


(a) Difference in #rules/flow in T_1.

(b) Difference in #rules/flow in T_2.



(c) Difference in hops in T_1.



(d) Difference in hops in T_2.

Fig. 6: Comparison with shortest alternative path routing.

of needed rules increased by about 18%. We observe similar behavior in T_2.

Fig. 5(c) shows the number of increased rules per flow for different maximum data rate of flows in T_1. We vary the maximum data rate of flows from 10 to 20 Mbps and keep the number of flows as 20. For all values of k the number of rules per flow increases with the increase of maximum data rate. Similar to the previous simulation, a higher value of k produces lower number of rules per flow. This is because when the max data rate is higher, the network observes more congested links. Because of a higher number of congested or soon to be congested links, there are less options for redirecting a flow. As a result, a higher number of new rules is needed to forward the flow. When the maximum data rate is 10 and $k = 1$ (or $k = 3$), the average number of needed rules to redirect a flow is 1.81 (or 1.44). The number of needed rules increased by about 20%. Fig. 5(d) plots the number of increased rules per flow for different maximum data rate of flows in T_2. We keep the same parameters as the previous simulation and observe the similar behavior. The overall number of rules /flow is higher in T_2 than T_1.

In Figs. 5(e) and 5(f), we show the number of increased rules per flow for different congestion threshold of links in T_1 and T_2. We vary the congestion threshold of links from 0.5 to 0.9 and keep the number of flows at 20. For all values of k, the number of rules per flow decreases with the increase of congestion threshold of links. This is because when the congestion threshold is higher, the network observes less congested links. Because of the lower number of congested or soon to be congested links, there are more options for redirecting a flow. As a result, the number of rules needed to forward the flow is smaller. When the congestion threshold is 0.5 and $k = 1$, the average number of needed rules to redirect a flow is 1.95. When $k = 3$, the average number of needed rules to redirect a flow is 1.56. The number of needed rules decreased by about 20%. We observe similar behavior in T_2.

Figs. 5(g) and 5(h) show the comparison between the grouping and merging based approaches in T_1 and T_2. We vary the number of flows from 20 to 100, keep the maximum data rate as 10 Mbps, and set $k = 1$. We observe that the merging based approach needs a smaller number of rules but a higher number of modifications than the grouping based approach. When the number of flows is 20 in T_1, the average number of needed rules to redirect a flow by grouping and merging based approaches are 1.69 and 1.67. The number of modification is needed in the merging based approach is 2.18 per flow. The merging based approach needs less (or more) rules when the number of flows is less (higher) than 50. The number of modifications for merging is always higher than grouping based approach. We observe similar behavior for T_2. Therefore, when we need to save storage on SDN switch, we need to use the merging based approach. When we prioritize interruption over SDN storage, then we need to use the grouping based approach.

Fig. 6 compares the alternate shortest path routing with redirection with k groups in terms of number of increased hops/flow and number of increased rules per flow. For these simulations, we vary the number of flows from 20 to 100 and keep the maximum data rate as 10 Mbps. Figs. 6(a) to 6(d) are executed with this settings. Fig. 6(a) shows the number of increased rules per flow for different number of flows in T_1. When the number of flows is 20 and $k = 1$, the average number of needed rules to redirect a flow in our approach is 0.58 less than the shortest alternate path approach. When $k = 2$ and $k = 3$, the average number of needed rules to redirect a flow in shortest alternate path approach is 0.62 and 0.57 higher than our approach, respectively. The difference is similar for all k values. The difference slightly decreases with the increase of number of flows. In Fig. 6(b), we present the number of increased rules per flow for different numbers of flows in T_2. The difference slightly increases then decreases with the increase of number of flows.

Fig. 6(c) shows the number of increased hops/flow for different numbers of flows in T_1. When the number of flows is 20 and $k = 1$, the average number of increased hops after redirecting a flow in our approach is 4.47 higher than the shortest alternate path approach. When $k = 2$ and $k = 3$, the average number of needed rules to redirect a flow in our approach is 4.66 and 4.35 higher than the shortest alternate path approach, respectively. The difference remains unchanged for all numbers of flows. Fig. 6(d) lists the number of increased hops/flow for different numbers of flows in T_2. We observe similar behavior as in T_1.

Therefore, based on the simulation result, we can conclude that our redirection approach with k links grouping can redirect by adding a lower number of rules than the shortest alternate path approach.

VII. Experiments in Datacenter

A. Experimental Settings

We conduct experiments to evaluate the effect on a flow when we change the routing. To conduct the experiments we use our small datacenter with fifteen SDN switches. We do not use any regular routers because they only increase hops without any effect on the changes in rules. The partial topology

of the datacenter is shown in Fig. 7(d). Nodes 2 to 16 are Pica 8 (P-3297) SDN switches. The numbers at the ends of a link denotes the port number where it is plugged. There are four servers (101-104) that are connected at the leaf level switches. We create three flows $\{101 \rightarrow 102, 101 \rightarrow 103, 101 \rightarrow 104\}$ and continuously record the ping delays (round trip delay). The initial flows travel the shortest path and we suddenly change the route by adding some rules. We compare the delays before, after, and during the rule change period. We use ONOS (2.2.2) as the controller and REST api to add/delete rules. We develop a separate Java program that can monitor the link status and add/delete rules using REST api.

B. Experimental Results

Fig. 7(a) shows the ping delay of 40 times of flow servers 101 to 102. The flow during time 1 to 16 follows the shortest path $\{101, 9, 10, 102\}$ and the average ping delay is 0.84 ms. After changing the rules by the rule modifier program, the flow travels the alternative path $\{101, 9, 5, 10, 102\}$. The flow during times 17 to 40 follows the alternative path and the average ping delay is 0.93 ms. The ping delay increases 0.09 ms for an increase of 1 hop. No other changes are observed for the addition of three new rules.

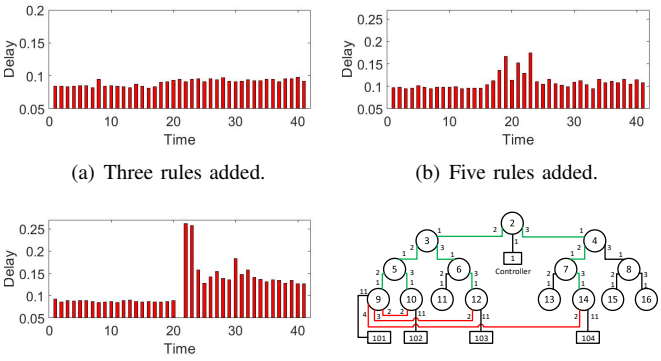
Fig. 7(b) depicts the ping delay of 40 times of flow from 101 to 103. The flow during times 1 to 14 follows the shortest path $\{101, 9, 12, 103\}$ and the average ping delay is 0.94 ms. After changing the rules, the flow travels the alternative path $\{101, 9, 5, 3, 6, 12, 103\}$. The flow during times 15 to 40 follows the alternative path. During time 15 to 23 we observe a lift in delay. This lift in delay occurs because of the addition of 5 new rules. The average ping delay becomes stable during times 24 to 40 and the average delay is 1.11 ms. The delay increases 0.17 ms for an increment of 3 hops.

Fig. 7(c) presents the ping delay of 40 times of flow servers 101 to 104. The flow during times 1 to 19 follows the shortest path $\{101, 9, 14, 104\}$ and the average ping delay is 0.87 ms. After changing the rules by the rule modifier program, the flow travels the alternative path $\{101, 9, 5, 3, 2, 4, 7, 14, 104\}$. The flow during times 20 to 40 follows the alternative path. At time 20 the destination remained unreachable for 1 s. During times 21 to 22 we observe a high lift in delay. This lift in delay occurs because of an addition of 7 new rules. The average average ping delay becomes stable during times 22 to 40 and the average delay is 1.38 ms. The ping delay increases 0.51 ms for an increase of 5 hops.

Therefore, we can conclude that the more the change in number of rules, the more the interruption in flow. Changes in up to 5 rules might not cause noticeable interruption to delay sensitive applications. Changes in 7 or more rules might cause interruption to delay sensitive applications.

VIII. Conclusion

The link flooding attack and link congestion are common issues in a datacenter. A larger number of rules plays an important role in transmission delay. When the number of rules in an SDN switch is higher than its fast accessible rules storage,



(a) Three rules added. (b) Five rules added.

(c) Seven rules added. (d) Datacenter topology (partial).

Fig. 7: Effects on ping delay.

the delay increases dramatically. Besides, changing rules in a higher number of SDN switches takes a long time, which causes interruption in flows. We consider this important issue for mitigating link flooding attack or congestion by redirecting some flows with less number of new rules. We propose and evaluate performances of k links flow grouping and merging-based approaches. Our extensive simulation supports that the proposed approaches can minimize the number of added rules by increasing the number of hops.

References

[1] 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.

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

[3] 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.

[4] 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.

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

[6] R. Biswas, J. Wu, W. Chang, and P. Ostovari, "Optimal filter assignment policy against transit-link distributed denial-of-service attack," in *2019 IEEE Global Communications Conference*, 2019, pp. 1–6.

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

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

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

[10] L. Dridi and M. F. Zhani, "Sdn-guard: Dos attacks mitigation in sdn networks," in *5th International Conference on Cloud Networking*, 2016.

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

[12] M. Sun, K. Shao, and L. Wang, "Minimizing flow rules for rerouting multi-flows in multi-failure recovery over sdn," in *8th International Conference on Software and Computer Applications*. Association for Computing Machinery, 2019, p. 555–559.

[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.