Optimal Monitor Placement Policy Against Distributed Denial-of-Service Attack in Datacenter

Rajorshi Biswas, Jie Wu, and Yang Chen
Department of Computer and Information Sciences
Temple University, Philadelphia, PA, USA.

***Abstract*—A distributed denial-of-service (DDoS) attack is a cyber-attack in which multiple attackers send out a huge number of requests to exhaust the capacity of a server, so that it can no longer serve incoming requests. In this paper, we propose a mechanism to protect against DDoS attacks originated within a datacenter. Our system is composed of two parts: flow monitoring and traffic filtering. In flow monitoring, we formulate two problems: one for finding flow assignments to monitors and another for selecting best locations of monitors. The first problem considers that the locations of monitors are predefined by the cloud provider and we provide an optimal solution. The second problem considers that the locations of monitors are not predetermined and there is a limit on the number of monitors. We propose a greedy solution for the second problem. The traffic filtering is trivial, as the DDoS flow can be blocked from the hypervisor of the source virtual machine. We present simulation results that strengthen support for our solutions.**

***Index Terms*—distributed denial-of-service, DDoS in datacenter, quality-of-service, software defined networking, network security.**

I. Introduction

A denial-of-service attack (DoS attack) is a cyber-attack in which the attacker seeks to make a machine or network resource temporarily unavailable to its users. CloudFlare is one of the biggest companies that provides DDoS protection services. CloudFlare users change the DNS of their domain and point to the CloudFlare DNS server. The CloudFlare DNS server returns a CloudFlare IP address instead of the user's IP address in DNS lookups. The server located at that IP address analyzes and filters the attack traffic. However, if the attackers remain in the same datacenter and know the public or private IP address of the victim, then the traffic sent to that IP address does not go through the CloudFlare server. Therefore, the internal DDoS traffic is not filtered by the CloudFlare server.

There are several methods to find the IP address of the server behind the CloudFlare protection server. Firstly, some websites give historical data about a domain, including IP address changes, sub-domains, and cloud providers. Secondly, if the target server runs any mail-server, then any email directly sent from the mail-server contains the IP address of the target server in its header. Generally, an email addressed to a wrong username (email) to a mail-server gets a reply that the username does not exist on the mail-server. Finally, a brute force approach can find the internal IP of the server. For example there are $16,777,216$ class A private IP addresses $(10.0.0.0 - 10.255.255.255)$. If one second is spent checking whether an IP address is the targeted server or not, a program with 194 threads can finish the checking in a day.

The security modules are expensive and they remain in the core or aggregation layer, which monitors the incoming (or outgoing) traffic from (or to) a datacenter. Therefore, an existing system cannot protect against internal DDoS attacks. Fig. 1 shows a scenario of an internal DoS attack. The victim V uses a commercial DDoS protection service. Therefore, the external attacker A_2 's traffic is blocked by the DDoS protection server. The internal attacker A_1 resides in the same datacenter as V. A_1's traffic does not go through the datacenter security devices. As a result, the attack traffic reaches V without any trouble. Other filter based DDoS attack defense mechanisms such as [1–3] cannot protect internal DDoS attacks. In order to prevent an internal DDoS attack, we need to do two things: monitor all the internal traffic and block the attack traffic. To block traffic, the controller creates a rule in the hypervisor firewall. Therefore, we focus on monitoring all internal traffic.

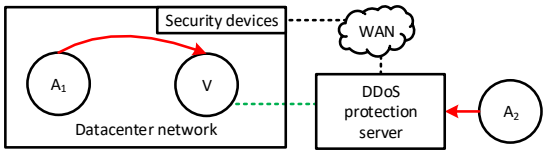


Fig. 1: Internal DDoS attack (A_1 and A_2: attacker, v: victim).

In this paper, we assume that the datacenter switches are all software defined networking (SDN) switches. Some of the free virtual machines (VMs) are used as traffic monitors. The monitors are special network function vitalization (NFV) units which can detect DDoS behavior of flows. The SDN switches probabilistically copy packets of each flow to a monitor. A monitor runs machine learning techniques to detect any DDoS behavior in the flows it receives. If a monitor classifies that a flow belongs to a DDoS attack, it creates a firewall rule at the hypervisor of source VM to stop the flow. The main contributions are as follows:

- We formulate a problem for assigning flows to monitors, considering that the location of the monitors are predefined. We provide an optimal solution by reducing the problem to a max-flow min-cost problem.
- We formulate another problem to optimally select some of the VMs among the free VMs for monitor placement, considering a budget on the number of VMs. This is an NP-hard problem and we provide a greedy solution.

The remainder of this paper is arranged as follows: Section II presents some related works. In Section III, we present the system model. Section IV and Section V present the formal definition of the first and second problem and our proposed optimal solutions, respectively. In Section VI, we present some simulation results. Finally, Section VII concludes the paper.

II. Related Work

There exist many statistical methods including correlation, entropy, covariance, cross-correlation, and information gain to detect anomalous DDoS requests [4]. A rank correlation-based method, Rank Correlation-based Detection (RCD), is proposed in [5]. An information theoretical approach using Kolmogorov complexity is used for the detection of DDoS attacks in [6]. A novel DDoS detection mechanism is proposed based on Artificial Neural Networks in [7]. There are other methods of detecting DDoS attacks, including [8, 9]. These methods can be used in monitors to detect DDoS attacks.

In [10], authors propose a three layer DDoS defence system, called Bohatei. It exploits the functionality of NFV/SDN architecture to mitigate distributed DoS attacks. There are some OpenFlow-based intrusion detection systems which utilize NFV/SDN functionalists [11, 12].

Authors in [13] propose a scheme to monitor the internal and external flows in a datacenter. The SDN switches mirror every flow to monitors. They use a greedy approach to find the locations for the monitors. At first, they find the districts that need at least one monitor. Then, they find assignments inside each district. Their approach sometimes fails to find an assignment due to the limited number of VMs. They also do not consider a budget on the number of VMs and fractionally copy packets of network flows. In [14], authors propose a two-stage DDoS mitigation framework. In the first step, it screens the traffic and determines what kind of processing is important for the flow, including network-layer security and application-layer security. In the second step, the advanced specialized detection system is used to detect DDoS behavior.

Large cloud providers such as Amazon EC2 and Microsoft Azure claim to provide protection against several traditional network security attacks, including DDoS. The DDoS mitigation techniques, including SYN cookies and connection limiting, are employed within EC2 [15]. Microsoft Azure is also capable of detecting internal DDoS attacks and remove the attacking VMs or accounts [16]. Though Azure and EC2 have mechanisms to protect against traditional DDoS attacks, they recommend the tenants to implement their own protection mechanism. Besides, many other datacenters are vulnerable to internal DDoS attacks. Authors in [17] discuss the detrimental effects by internal attacks including low rate DDoS attack.

The aforementioned works that exploit the benefit, of SDN and NFV either do not consider the network overhead and limited budget on number of VMs, or they partially process the traffic flows. Therefore, it is necessary for a new system to consider a limited budget on the number of VMs and aim to partially or fully monitor all the network flows.

III. System Model

A. Datacenter Model

Our system is composed of physical machines (PM), top of rack (TOR) switches, SDN switches, users, attackers, and a victim. The system model is depicted in Fig. 2. The victim (v) uses a DDoS protection service from a commercial

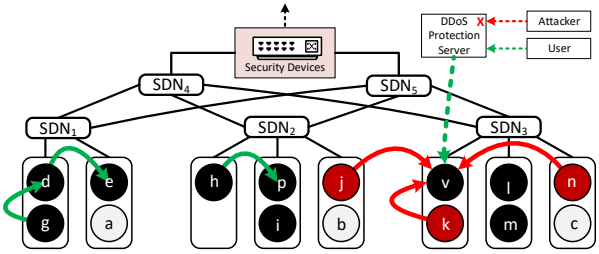


Fig. 2: System model.

DDoS protection service provider, which is located outside the datacenter. The top level of the datacenter is equipped with flow monitoring devices which monitor incoming and outgoing traffic to/from the datacenter. Any external attacker or user (that remains outside the datacenter) has to go through the DDoS protection server and the datacenter security module. Therefore, the external attackers are easily blocked by either the DDoS protection server or the datacenter security module. The traffic from an internal attacker/user does not go through the security module or the commercial DDoS protection server.

The SDN controller controls all the flow monitoring process. It deploys monitors to some free VMs. Then, it assigns each flow to a monitor and the monitor analyzes the packets in the flow to detect whether it belongs to an attacker or not. There are many methods, including [8, 9], to classify whether packets in a flow belong to an attacker, discussed in Section II. If the flow belongs to an attacker then the monitor notifies the controller. The controller creates a firewall rule in the originating VM's hypervisor to block the flow. Assigning a flow to a monitor happens in two steps. In the first step, the SDN switch on the flow path closest to the monitor is selected. In the second step, an SDN or hypervisor rule is created to copy the packets and forward them to the monitor. Let us assume that the flow f_{jv} (flow from VM j to v) is assigned to the monitor installed in VM c. The packets of flow f_{jv} can be copied and forwarded to c from any of the SDN switches in $\{SDN_2, SDN_5, SDN_3\}$. SDN_3 is closest to c. Therefore, the controller selects SDN_3, copies, and forwards the packets to c. The reason for selecting the closest SDN switch is that it increases the minimum network overhead for copying a flow. We also consider the limited capability of monitors. We assume that each monitor can handle a limited number of flows. Though in reailty, capability of each VM can be different. For simplicity, we assume that all VMs are homogeneous with the same capabilities. Besides, we can easily extend the solution for homogeneous VMs to the solution for heterogeneous VMs.

When the number of required monitors for all the flows is less than the budget, the controller creates rules to probabilistically copy a certain portion of each flow from the SDN switch. For example, assume there are 2 available VMs for monitors and each monitor can handle a flow. If the number of flows is 4, then each of the monitors needs to handle 2 flows. Therefore, the SDN switches copy 50% of the packets of each flow and forward them to the monitors. The copy of the original packets creates extra network overhead. We measure the increased network overhead by multiplying the total data

rate by the number of hops the flow copy travels. Let $f_{ab} \in F$ be a flow between VM a and b. r_{ab} is the data rate of flow f_{ab}. V is the set of free VMs and $A : F \rightarrow V$ is the monitor assignment. The network overhead for assignment A is $C(A)$.

$$C(A) = \sum_{f_{ab} \in F} r_{ab} \times \min_{p \in a \rightarrow b} dist(p, A(f_{ab})) \quad (1)$$

Here, $a \rightarrow b$ is an ordered set of SDN switches on the path from a to b, including the physical machines of a and b. $dist(p, A(f_{ab}))$ is the number of hops from SDN switch p to the monitor $A(f_{ab})$. For example, if the data rate of the flow f_{jv} is 1 (Mbps) and it is assigned to monitor c, then the increased network overhead for monitoring the flow is $1 \times 1 = 1$. If the flow f_{jv} is assigned to monitor b, then there is no network overhead. This is because, the hypervisor can copy the flow to the monitor from the source VM of flow f_{jv}. So, the copied packets do not travel any links. The increased network overhead can be a reason to degrade QoS of the datacenter. Therefore, our main goal is to minimize the increased network overhead. To minimize the network overhead, we need to select the best subset of the free VMs for the monitor placement and find the best flow assignment. The flows in a datacenter is not static. Therefore, the controller needs to re-calculate flow assignments periodically.

B. Attack Model

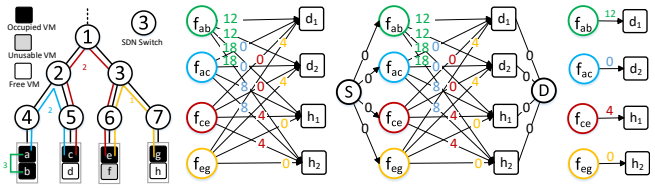
We assume all the attackers are internal attackers and controlled by a master. The master may not reside inside the datacenter. It is capable of finding the actual IP (public or private) address of the victim. When the master sends commands telling the attackers the victim's IP address, they start sending attack packets. The victim becomes overwhelmed with the attack packets and DoS occurs. For example, in Fig. 2, VMs j, k, and n are the attackers.

IV. Flow Assignment

In this section, we formulate the problem to find an optimal flow assignment to the monitors. We consider that the cloud provider allocates some VMs for monitor deployment. The number of VMs, location, and type of VMs are decided based on affordability and the business strategy of the cloud provider. For example, the cloud provider may not want to allocate VMs in a PM, which is highly demanded by customers.

A. *Problem 1: Find a flow assignment so that the network overhead is the minimum by ensuring coverage of all internal flows.*

Let $f_{ab} \in F$ be the flow between VM_a and VM_b ($VM_a \in V$ and $VM_b \in V$). The data rate of the flow f_{ab} is r_{ab}. There are M ($|F| = M$) number of flows and N ($|V| = N$) number of free VMs. The cloud provider can afford VMs in V' ($|V'| = K$) from some predefined locations. Therefore, each VM needs to monitor M/K flows. We need to find a flow assignment $A : F \rightarrow V'$ for which the increased network overhead is minimum. The problem can be expressed as the following:



(a) G (b) G' (c) G'' (d) A

Fig. 3: Problem I example.

Algorithm 1 Flow Assignment

Input: The topology G, set of flows F, and set of VMs V'.
Output: A flow assignment.
1: **Procedure:** FLOW-ASSIGNMENT(G, F, V')
2: $G' \leftarrow$ CREATE-BIPARTITE (G, F, V')
3: $G'' \leftarrow$ CREATE-FLOW-GRAPH (G')
4: $A \leftarrow$ find assignment using max-flow min-cost in G''.
5: **return** A.

$$\begin{aligned} \text{minimize} \quad & \sum_{f_{ab} \in F} r_{ab} \times \min_{p \in a \rightarrow b} dist(p, A(f_{ab})) \\ \text{subject to} \quad & \forall_{v \in V} |\{f : f \in F \text{ and } A(f) = v\}| \end{aligned} \quad (2)$$

Here, p is the node on the path $a \rightarrow b$ and $dist(p, A(f))$ is the number of hops from the node p to the VM that monitors the flow f_{ab}. The location of K VMs are predetermined. Therefore, we only need to find an optimal assignment between the flows and free VMs. Usually, datacenters use regular switches with SDN switches which are not programmable and cannot copy packets. If a flow does not go through any SDN switches, then the hypervisor of source or destination is the only way of copying the flow. For simplicity, we consider all the switches in our model to be SDN switches.

B. An Optimal Flow Assignment Scheme

The assignment process consists of two steps. In the first step, we construct a bipartite graph $G' = (V_1, V_2, E)$, where V_1 is the set of flows ($|V_1| = M$) and V_2 is the set of parts of VMs. Each VM is split into $VM_1, VM_2, ..., VM_B$ parts so that each part can handle one flow ($|V_2| = B \times K$). From each flow, edges are added to all parts. The weight of the edge is the increased network overhead. So, G' is a complete bipartite graph ($K_{M,BK}$).

In the second step, we find a minimum cost perfect matching. We can easily reduce the problem to a minimum cost flow problem. A flow graph $G'' = (V'', E'')$ is created from G' ($V'' = V_1 \cup V_2$ and $E'' = E$). A source and a destination are added to V''. Then, edges from the source to all of V_1 and the destination to all of V_2 are added to E''. The capacity of each edge/link is 1 and the cost of flow is the weight of each edge. Newly added edges will have zero cost. The max-flow min-cost paths in G'' produce a minimum cost perfect matching in G'. We can find max-flow min-cost paths using any of the algorithms in [18]. The complete algorithm is shown in Alg. 1. Procedures CREATE-BIPARTITE and CREATE-FLOW-GRAPH are not shown in details to save space.

Let us consider the simple topology in Fig. 3(a). There are seven SDN switches, each connected to a PM. Each PM can

host two VMs. There are four flows f_{ab}, f_{ac}, f_{ce}, and f_{eg}. The data rates of these flows are $r_{ab} = 3$, $r_{ac} = 2$, $r_{ce} = 2$, and $r_{eg} = 1$. There are three free VMs $\{d, f, h\}$. The cloud provider can only afford d and h. We want to find a mapping from the four flows to two VMs. Therefore, each VM will handle two flows.

According to the first step, we create a bipartite graph $G' = (V_1, V_2, E)$ where $V_1 = \{f_{ab}, f_{ac}, f_{ce}, f_{eg}\}$. We split d into d_1, d_2 and h into h_1, h_2. So, $V_2 = \{d_1, d_2, h_1, h_2\}$. Then we add edges between all pairs of nodes from V_1 and V_2. The weight of each edge is the increased network overhead for monitoring it. For example, the weight of the edge between f_{ab} and h_1 is 12. If we want to monitor f_{ab} in d, then the hypervisor needs to copy the flow and the flow travels $4 \rightarrow 2 \rightarrow 5 \rightarrow d$, which is four hops. Therefore, the increased network overhead is $4 \times 3 = 12$. Similarly, the weight of the edge between f_{ab} and h_2 is also 12. If we want to monitor f_{ac} in d, then the hypervisor of d and c can copy the flow to d and there is no additional network overhead. Therefore, the weight of the edge between f_{ac} and d_1 (or d_2) is 0.

According to the second step, we create the flow graph $G'' = (V'', E'')$. We add S and D to G'' ($V'' = V_1 \cup V_2 \cup \{S, D\}$). After adding all edges in E, edges from S to each node in V_1 and each node in V_2 to D are created and added to E''. Weights of newly created edges are zero. We assign the capacity of every edge to one. Fig. 3(c) shows the flow graph. Then we use one of the maximum flow minimum cost algorithms [18] to find a perfect matching between V_1 and V_2. Fig. 3(d) shows the perfect matching (A). Therefore, f_{ab} and f_{ac} will be monitored in VM d, and f_{ce} and f_{eg} will be monitored in VM h. The total network overhead for monitoring these four flows is 12+4=16.

Theorem 1. *The complexity of Alg. 1 is $O(M^3 + S^3)$.*

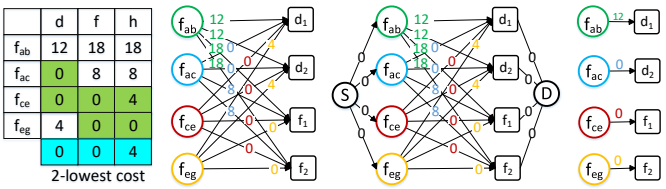
Proof. In step 2 of Alg. 1, creating a bipartite graph takes $O(M^2)$ if we pre-compute all pairs' shortest path distances. Computation of all pairs' shortest path distances takes $O(S^3)$, where S is the number of SDN switches. Then, in step 2, creating the flow graph takes $O(M)$. In step 3, finding the assignments using the cheapest augmenting path algorithm takes $O(M^3 + M^2 log(M))$ time, which is $O(M^3)$. Therefore, the complexity of the Alg. 1 is $O(M^3 + S^3)$. □

Theorem 2. *The Alg. 1 returns optimal flow assignment.*

Proof. The capacity of all edges in E'' is 1. Therefore, for any assignment, the maximum flow is M. To ensure M amount of flows between S and D, each node in V_1 must have an incoming flow of 1. Similarly, each node in V_2 must have an outgoing flow of 1. Therefore, no node in V_1 is assigned to multiple nodes in V_2. Secondly, the minimum cost maximum flow algorithms [18] return the minimum cost path in G''. Therefore, Alg. 1 returns the optimal flow assignment. □

V. Monitor Placement

In this section, we formulate a problem to find optimal locations for the monitors among some free VMs. We consider a limited budget on the number of VMs for allocating monitors. The large number of running VMs in the datacenter increases the energy consumption and cost. The cloud provider may want to reserve some VMs for a sudden increase in the customers' demand. Therefore, a cloud provider wants to deploy monitors on the limited number of VMs. At the same time, the cloud provider wants to get maximum security from the monitors and less network overhead.



(a) CM (b) G' (c) G'' (d) A

Fig. 4: Problem II example.

Algorithm 2 Flow Assignment 2

Input: The topology G, set of flows F, and set of VMs V'.
Output: A flow assignment
1: **Procedure:** FLOW-ASSIGNMENT-2(V, K)
2: Calculate M/K-lowest cost for each VM in V.
3: Select K lowest M/K-lowest cost VMs V' from V.
4: $A \leftarrow$ FLOW-ASSIGNMENT(G, F, V')
5: **return** A.

A. *Problem 2: Find locations for monitors and a flow assignment so that the network overhead is the minimum by ensuring coverage of all the internal flows.*

In this problem, we assume that the locations of K VMs for installing monitors are not predetermined. We need to find a flow assignment $A : F \rightarrow V'$ and find V' ($V' \subset V$) for which the network overhead is minimum. The problem can be expressed as the following:

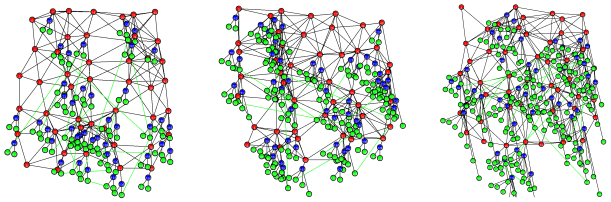
$$\begin{aligned} \text{minimize} \quad & \sum_{f_{ab} \in F} r_{ab} \times \min_{p \in a \rightarrow b} dist(p, A(f_{ab})) \\ \text{subject to} \quad & |V'| = K, \\ & \forall_{v \in V'} |\{f : f \in F \text{ and } A(f) = v\}| \end{aligned} \tag{3}$$

So, we need to find an optimal assignment of the flows to the monitors for which the network overhead is the minimum. The problem 2 is NP-hard. Its NP-hardness can be proved by reducing it to the vertex cover problem [13].

B. Greedy Approximation: M/K-lowest cost approach

M/K is the number of flows to be monitored in a VM. M/K-lowest cost of a VM is the lowest total network overhead for assigning M/K flows to the VM. Inclusion of a VM to the assignment set increases at least M/K-lowest cost. Therefore, we select K VMs with the lowest M/K-lowest cost. We denote the M/K-lowest cost of VM a by C'_a. The complete algorithm is shown in Alg. 2. Next flows are assigned to the selected VMs using Alg. 1.

Let us consider the example in Fig. 3(a). Here, $M = 4$ and $K = 2$. We need to find the 2-lowest cost for each VM. The costs for monitoring f_{ab}, f_{ac}, f_{ce}, and f_{eg} in VM d are 12, 0, 0, and 4, respectively. The 2-lowest cost of VM d is 0. Similarly, the 2-lowest costs of h and f are 4 and 0, respectively. The



(a) Topology 1. (b) Topology 2. (c) Topology 3.

Fig. 5: Randomly generated topologies.

VMs with lowest 2-lowest costs are f and d. Therefore, $\{f, d\}$ is the set of selected K VMs. The cost matrix (CM), G', G'', and A are shown in Fig. 4.

Next, we create a bipartite graph $G' = (V_1, V_2, E)$. The $V_1 = \{f_{ab}, f_{ac}, f_{ce}, f_{eg}\}$ and $V_2 = \{d_1, d_2, f_1, f_2\}$. Then, we add edges between all pairs of nodes from V_1 and V_2, and assign weights to all edges. Fig. 4(b) shows the bipartite graph. Then, we create the flow graph $G'' = (V'', E)$ by adding S and D to G''. Fig. 4(c) shows the flow graph. Then, we use one of the maximum flow minimum cost algorithms to find a perfect matching between V_1 and V_2. Fig. 4(d) shows the perfect matching. Therefore, f_{ab} and f_{ac} will be monitored in VM d, and f_{ce} and f_{eg} will be monitored in VM f. Fig. 4(d) shows the final flow assignment. The total network overhead for monitoring these four flows is 12+0=12. The final flow assignment is shown in Fig. 4(d).

Theorem 3. *The complexity of Alg. 2 is $O(M^3 + S^3)$.*

Proof. In step 2, calculating the M/K -lowest cost takes $O(MN)$. In step 3, finding K lowest M/K-cost takes $O(N)$. Step 4 which is finding an assignment using Alg. 1 takes $O(M^3+S^3)$. So, the complexity of Alg. 1 is $O(M^3+S^3)$. □

Theorem 4. *The M/K-lowest cost approach increases at most $2R \log S$, where R is total bandwidth of the flows.*

Proof. Let the optimal network overhead be denoted by C^*. The cost approximated by the M/K-lowest cost approach is C. The set V' ($|V'| = K$) has the lowest M/K-lowest cost among the V VMs. V^* is the optimal set of VMs. In the worst case, $V - V'$ and $V - V^*$ are disjoint. Therefore, we can write

$$C^* = C - 2\log(S) \sum_{a \in V - V'} R_a + \sum_{b \in V - V^*} C'_b \tag{4}$$

Here, R_a is the total incoming data rate to VM a and C'_b is M/K-lowest cost of VM b. In the worst case, $\sum_{b \in V - V^*} C'_b$ is zero. Therefore, we can write the following:

$$\begin{aligned} C^* &= C - 2\log(S) \sum_{a \in V - V'} R_a, \\ C^* &\geq C - 2\log(S) \sum_{a \in V} R_a, \quad C \quad \leq C^* + 2\log(S)R \end{aligned} \tag{5}$$

So, C will be at most $2R\log(S)$ more than the optimal cost. In tree topology, $2\log(S)$ is the diameter. Therefore, for any kind of topology the increased network overhead will be at most $R \times diameter$ more than the optimal. □

TABLE I: Topology Parameters

Number of	Topology 1	Topology 2	Topology 3
Nodes	172	249	277
VMs	84	150	184
PMs	42	52	44
SDN switches	46	47	49
Links	304	392	427
PMs in an SDN switch	{1}	{1, 2}	{1, 2}
VMs in a PM	{2}	{2, 3, 4}	{1, 2, 3, 4, 5, 6, 7}
Datarate	[1,6]	[1,6]	[1,6]

TABLE II: Random tree topology settings

Number of	Settings 1	Settings 2	Settings 3
Max degree	3	5	8
PMs in an SDN switch	3	3	3
VMs in a PM	5	5	5

VI. Experimental Results

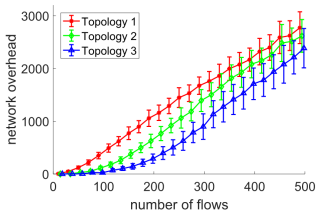
A. Experimental Settings

We conduct the experiments with a custom build java simulator. The main reason for using a custom build for the simulator is its scalability. We only need to count the increased network overhead for monitoring all flows. The network topologies we consider contain $200-300$ SDN switches, PMs, and VMs. Using NS3 or other similar simulators for this kind of simulation would take several days. That is why we built our own java multi-threaded simulator to get the results quickly.

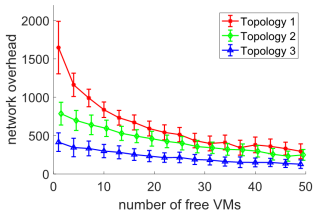
We generate three random topologies for some experiments (in Fig. 6). We consider a 500×500 rectangular region. The region is divided into 50×50 blocks. In each block, an SDN switch is placed by choosing a random location uniformly. Links between two SDN switches are added if their distance is less than 100 units. SDN switches in Topologies 1, 2, and 3 are generated with this setting. Then, PMs are attached to each SDN switch with probability of 0.5. The number of PMs attached to an SDN switch is 1 in Topology 1. The number of PMs attached to an SDN switch is taken randomly from $\{1, 2\}$ in Topologies 2 and 3. Then VMs are added to each PM. The number of VMs is taken randomly from $\{2\}$, $\{2, 3, 4\}$, and $\{1, 2, 3, 4, 5, 6, 7\}$ in Topologies 1, 2, and 3. Node colors red, blue, and green represent SDN switches, PMs, and VMs, respectively. Detailed information is listed in Table I.

We conduct some experiments with tree datacenter topologies. In reality, many datacenters use the tree structure. We generate random trees with three different settings. We first generate the desired number of nodes and randomly pick a root among the nodes. Then, a random node from the generated nodes is picked up and added as a child to a random node in the tree. Then, at each leaf node, PMs and VMs are added similarly. We define three different settings for the random tree generation. All settings allocate 3 PMs in each leaf node and 5 VMs in each PM. The maximum node degrees in Setting 1, 2, and 3 are 3, 5, and 8, respectively.

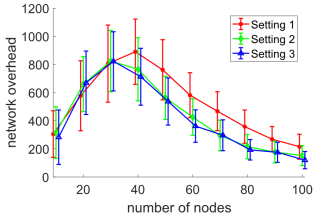
To generate flows, we consider two parameters: the number of free VMs, and the number of flows. Firstly, the desired number of VMs is excluded from the VM list. The refined VMs are used as the source and destination of a flow. Then, all possible pairs of VMs are generated. From the pairs of



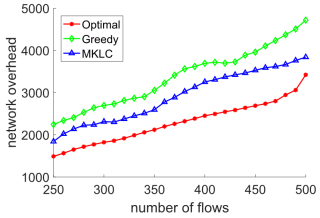
(a) Effect of the number of flows.



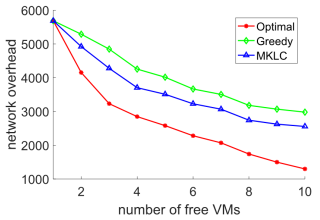
(b) Effect of the number of free VMs.



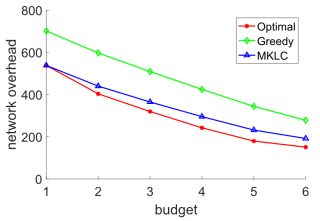
(c) Effect of the number of nodes.



(d) Effect of the number of flows.



(e) Effect of the number of free VMs.



(f) Effect of the number of nodes.

Fig. 6: Simulation results.

VMs, the desired number of pairs are chosen randomly and a flow is created between the VMs in each pair. The data rate of each flow is chosen randomly from the range $[1, 6]$.

We record the increased network overhead for 100 samples and plot the average values. Since the solution for problem 1 is optimal, we compare the network overhead for different settings by varying different parameters. For the second problem, we compare the optimal, M/K-lowest cost, and greedy approachs' network overhead. To find the optimal solution, we needed to check all combination of free VMs. Therefore, we pick a small number for free VMs and budget, then observe the effect of changing multiple parameters. The greedy approach is mainly introduced in [13] for intra-district flow assignments in a datacenter. We consider a slightly modified greedy approach compared with the original. For each flow, we pick the best VM and add it to the selected VMs list. We keep adding VMs to the selected VMs list unless the budget is met. When the number of selected VMs is equal to the budget, the new flow selects the best VM among the selected VMs. When a VM's capacity becomes full, we do not consider the VM for the next flow. Thus, we assign VMs for each flow and the selected VMs are used for monitor placement.

B. Simulation Results

We compare the increased network overhead of three topologies. In Topology 1, we observe a small number of PMs. Topologies 2 and 3 have a higher number of PMs than Topology 1. The number of VMs in Topology 3 is the highest, Topology 1 is lowest, and Topology 2 is in between. Therefore, it is expected that Topology 1 has a higher network overhead. Though Topology 2 has a higher number of PMs, it does not help to reduce overhead. This is because VMs in the same PM do not increase network overhead between them, but VMs under the same SDN switch increase network overhead.

Fig. 6(a) shows the average network overhead and the standard deviation for different number of flows. We vary the number of flows from 1 to 500. We keep the minimum number of free VMs as 20 for all topologies. When the numbers of flows are small, the number of free VMs is higher than 20. In Topology 1, if the number of flows is less than 32 $((84-20)/2)$, there might be more than 20 free VMs. If the number of flows is greater than 32, it does not guarantee that the number of VMs is 20. The higher the number of flows, the higher the probability of having exactly 20 free VMs. When the number of flows is 500, the increased network overheads for Topologies 1, 2, and 3 are $2,772$, $2,617$, and $2,385$. The standard deviation of the network overhead for Topologies 1, 2, and 3 are 302, 310, and 372, respectively.

Fig. 6(b) shows the average network overhead and standard deviation for different numbers of free VMs. We vary the number of free VMs from 1 to 50. We keep the number of flows as 500 for all topologies. When the number of free VMs is smaller, the network overhead of different topologies is higher. This is because in Topology 1, there are a small number of VMs and less flexibility of placing the monitors. As a result, the network overhead is higher. The network overhead decreases with the number of free VMs.When the number of free VMs is higher the difference of the network overhead is smaller. The standard deviation of network overhead is also decreased by the number of free VMs. When the number of free VMs is 1, the network overhead for Topologies 1, 2, and 3 are 1646, 785, and 414, respectively. When the number of free VMs is 50, the network overhead for Topologies 1, 2, and 3 are 295, 248, and 128, respectively.

Fig. 6(c) shows the average overhead and the standard deviation for different numbers of SDN switches. We vary the number of SDN switches from 10 to 100. We use the random tree generation algorithm, as discussed in experimental settings. We record the network overhead for 100 randomly generated trees with 500 flows. The settings 1 trees are supposed to have a greater height than settings 2 or 3 trees. So, the network overhead is supposed to be higher in topologies with smaller node degrees. The maximum number of flows is set to 500 for all settings. We keep the number of free VMs equal to 50% of the number of SDN switches. We observe that there is no significant difference in network overhead when the number of SDN switches ranges from 10 to 30. This is because the maximum allowable flows almost saturate the network. When the number of SDN switches is greater than 30, we see that settings 1 topologies have a higher network overhead than settings 2 and 3. A datacenter with moderate flows and a higher node degree helps to reduce network overhead. We also observe that the overhead is increasing by the number of SDN switches till 30. This is because the maximum possible

flows do not cross 500 in the topologies. When the number of SDN switches increases, the number of VMs and the maximum possible flows also increase. When the number of SDN switches is more than 30, the number of flows in the network remains constant (500). The higher the number of SDN switches, the higher the number of VMs, free VMs, and flexibility. As a result, the network overhead decreases with the number of SDN switches.

We compare the performances of the optimal (brute force), greedy, and M/K-lowest cost (MKLC) approaches by varying different parameters. Fig. 6(d) shows the average network overhead of optimal, greedy, and MKLC for different numbers of flows. We varied the number of flows from 250 to 500. We keep the number of SDN switches to be 100. We choose the range because they neither saturate the network, nor are very small for the network. We use the settings 3 and the number of flows as 500 for all topologies. We keep the budget constant (9 VMs) to see the performance of three approaches and their effect on the number of flows. The overhead for greedy is higher than MKLC. On average, the network overhead is $3,366$, $2,913$, and $2,279$ for greedy, MKLC, and optimal approaches, respectively. The MKLC increases the network overhead by 27% while the greedy increases overhead by 48% more than the optimal network overhead.

Fig. 6(e) shows the average network overhead of the three approaches for different numbers of free VMs flows. We vary the number of free VMs from 1 to 10. We keep the same settings as the previous experiment for the topology generation. We set the budget to be 50% of the number of free VMs. When there is a free VM, all approaches will assign all flows to that VM. Therefore, all approaches have the same network overhead. When the number of free VMs is 2, the greedy approach is more likely to not choose the best VMs for placing monitors than the MKLC approach. A higher number of free VMs gives more options and a higher location flexibility. As a result, the network overhead decreases by the number of VMs. We observe that the MKLC always performs better than the greedy approach. On average, the network overhead is $4,049$, $3,607$, and $2,507$ for greedy, MKLC, and optimal approaches, respectively.

Fig. 6(f) shows the average network overhead of the three approaches for different budgets. We vary the budget from 1 to 6. We keep the number of SDN switches as 20. We use the setting 3 and number of flows as 100 for all topologies. It is clearly observed that the overhead decreases by the budget. The network overhead in MKLC approach is closer to the optimal than the greedy approach. On average, the network overheads are 470, 336, and 291, respectively. Therefore, we can conclude from the above experiments that, the MKLC approach produces less overhead than the greedy approach.

VII. Conclusion

The internal DDoS attack is less common than the regular DDoS attack, and is also harder to detect and protect against. SDN switches can be utilized to monitor internal network flows. Besides, the regular SDN switches can be used to copy network packets to monitor by developing a custom action plugin. The OpenFlow framework supports custom plugins, which can be used for probabilistic packet forwarding. In this work, we present two problems for assigning flows to monitors and selecting the best locations for the monitors. We compare the performances of the proposed placement policy with an existing greedy approach. Simulation results show that our proposed approach works better than the greedy approach. In the future, we may analyze the performance of detection methods for different probabilities of packet copying.

Acknowledgments

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1618398, CNS 1651947, and CNS 1564128.

References

[1] R. Biswas and J. Wu, "Filter assignment policy against distributed denial-of-service attack," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems*, Dec 2018.

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

[3] R. Biswas, J. Wu, and A. Srinivasan, "Cost-aware optimal filter assignment policyagainst distributed denial-of-service attack," in *Resilience Week*, Nov 2019.

[4] 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.

[5] W. Wei, F. Chen, Y. Xia, and G. Jin, "A Rank Correlation Based Detection against Distributed Reflection DoS Attacks," *IEEE Communications Letters*, vol. 17, no. 1, Jan 2013.

[6] A. Kulkarni and S. Bush, "Detecting Distributed Denial-of-Service Attacks Using Kolmogorov Complexity Metrics," *J. Netw. Syst. Manage.*, vol. 14, no. 1, Mar. 2006.

[7] T. A. Ahanger, "An effective approach of detecting DDoS using Artificial Neural Networks," in *2017 International Conference on Wireless Communications, Signal Processing and Networking*, Mar 2017.

[8] X. Ma and Y. Chen, "DDoS Detection Method Based on Chaos Analysis of Network Traffic Entropy," *IEEE Communications Letters*, vol. 18, no. 1, Jan 2014.

[9] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A System for Denial-of-Service Attack Detection Based on Multivariate Correlation Analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, Feb 2014.

[10] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and Elastic DDoS Defense," in *USENIX Security Symposium*, 2015.

[11] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, "Let SDN Be Your Eyes: Secure Forensics in Data Center Networks," in *Workshop on Security of Emerging Networking Technologies*, Jan 2014.

[12] M. A. Lopez and O. C. M. B. Duarte, "Providing elasticity to intrusion detection systems in virtualized Software Defined Networks," in *2015 IEEE International Conference on Communications*, Jun 2015.

[13] P. Lin, C. Wu, and P. Shih, "Optimal Placement of Network Security Monitoring Functions in NFV-Enabled Data Centers," in *2017 IEEE 7th International Symposium on Cloud and Service Computing*, Nov 2017.

[14] T. Alharbi, A. Aljuhani, and H. Liu, "Holistic DDoS mitigation using NFV," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference*, Jan 2017.

[15] Amazon, "Amazon web services: Overview of security processes," May 2017.

[16] A. Marshall, M. Howard, G. Bugher, B. Harden, C. Kaufman, M. Rues, and V. Bertocci, "Security best practices for developing windows azure applications," *Microsoft Corp*, Jun 2010.

[17] N. Jain and S. Kamara, "Attacking data center networks from the inside," Jun 2015.

[18] R. K. Ahuja, *Network Flows: Theory, Algorithms, and Applications*, 1st ed. Pearson Education, 2017.