Agile Virtualized Infrastructure to Proactively Defend Against Cyber Attacks

Fida Gillani*, Ehab Al-Shaer*, Samantha Lo†, Qi Duan*, Mostafa Ammar† and Ellen Zegura†

*University of North Carolina Charlotte (UNCC) {sgillan4, ealshaer, qduan}@uncc.edu
†School of Computer Science, Georgia Institute of Technology, {samantha, ammar, ewz}@cc.gatech.edu

Abstract—The DDoS attack has been a persistent threat to network availability for many years. Most of the existing mitigation techniques attempt to protect against DDoS by filtering out attack traffic. However, as the critical network resources are usually static, the adversaries are able to bypass filtering by sending stealthy low traffic from large number of bots that mimic benign traffic behavior. Sophisticated stealthy attacks on critical links can cause a devastating effect such as partitioning domains and networks. In this paper, we propose to defend against DDoS attack by proactively changing the footprint of critical resources in an unpredictable fashion to invalidate the adversary’s knowledge and planning attack against critical network resources. Our present approach employs virtual networks (VNs) to dynamically reallocate network resources using VN placement and offers constant VN migration to new resources. Our approach has two components: (1) a correct-by-construction VN migration planning that significantly increases the uncertainty about critical links of multiple VNs while preserving the VN placement properties, and (2) an efficient VN migration mechanism that identifies the appropriate configuration sequence to enable node migration while maintaining the network integrity (e.g., avoiding session disconnection). We formulate and implement this framework using SMT logic. We also demonstrate the effectiveness of our implemented framework on both PlanetLab and Mininet-based experimentations.

I. INTRODUCTION

Network robustness is essential for service availability and quality of service against increasingly common cyber threats, such as DDoS [1]. The indispensable nature of these services for today’s world make them mission critical and any disruption to them could be catastrophic. The DDoS attack can be either direct or indirect. In direct DDoS, the attack traffic is sent directly to the victim destinations to flood the last mile link. In indirect DDoS, the attack traffic is sent to the geographical neighbors of the victim destinations to flood critical links in the network shared between the neighbors and the victims. We concern ourselves with indirect DDoS attacks. Existing DDoS mitigation techniques [2], [3] try to defend by filtering out attack traffic. However, as the attacks are going more stealthy (like sending low traffic per bot to mimic normal user) [1], distinguishing benign from attack is not feasible in most cases. Nonetheless, a common prerequisite of all link-based DDoS attacks is to identify critical target(s) through reconnaissance (e.g., small set of links carrying most

This research was supported in part by National Science Foundation under Grant No. CNS-1320662. Any opinions, findings, conclusions or recommendations stated in this material are those of the authors and do not necessarily reflect the views of the funding sources.
the critical only migration should take place to evade the attack successfully. This is our second contribution in the paper.

The purpose of agility is to frequently replace potentially discovered critical substrate with valid new threat safe resources such that (1) the selection process of the new resources is unpredictable by the adversary, and (2) the migration to these new resources will be faster than the reconnaissance time of the adversary. This, therefore, will constantly invalidate the adversary assumptions about the VN placement, discovered in the reconnaissance phase and this is our third contribution. All these constraints are formally defined as a part of the agile VN framework model using SMT [13]. Advanced SMT solvers such as Z3 [14] and Yices [15] can solve tens of thousands of constraints and millions of variables [13]. Therefore, our approach is scalable to large networks with multiple VNs.

The fourth contribution is in developing a migration mechanism that executes VN migration strategy generated by the framework on a virtualized physical network. It handles all the logistics of the movement including the exact sequence/schedule of the steps to be executed in order to complete the move and the timing of such steps. We use PlanetLab [16] as our virtualized infrastructure and extend the functionality of an existing PlanetLab based controller, PL-VNM [17], to implement our migration mechanism. We use this PlanetLab implementation to rigorously evaluate the effectiveness of this framework. We also use simulation and Mininet based experiments to evaluate other features (like scalability) which were not possible on Planetlab.

Our proposed solution only requires provision in the network to have redundant paths that can be used to switch critical role to non-critical. Furthermore, the proposed framework model can be extended by adding more functional, security or threat related constraints.

In the rest of the paper, we explain related work in Section II followed by our approach in Section III. Threat modeling is explained in Section IV followed by agile migration in Section V. The migration mechanism and evaluation are presented in Section VI and Section VII, respectively.

II. RELATED WORK

The existing DDoS remedies, all requiring the ability to distinguish attack traffic from benign, can be divided into two major categories, infrastructure based and virtual network based approaches. Some infrastructure based approaches proposed adding authentication bits in packet headers to assist routers drop attack traffic [18]–[20]. Others, suggested routers to signal each other through a signaling protocol to filter DDoS traffic [21]–[23]. The major limitation of these approaches is to expect major changes in existing hardware and network protocols, which is hard to come by. The VN based approaches [2], [24]–[28] suggest deploying a secret virtual layer on top of the substrate to route traffic only through designated routers. For example Akamai’s SiteShield service [3], SOS [2], [28] and Mayday [24] use a shared secret (e.g., IP of the target) to route traffic through specialized routers. The major shortcoming with such approach is that if the secret (virtual layer) is revealed then the adversary can bypass the virtual network to directly flood the target. Some countermeasure approaches [2], [24] were also proposed preventing malicious overlay components to disclose shared secret, but they require expensive mechanisms like anonymous routing, etc. But, none of these approaches can counter the Crossfire attack [1] because it does not directly attack the target.

On the VN agility front, the work done in [7] is primarily concerned with one time VN placement but allows a limited moving of ensemble of VNs deployed on the substrate under resource contention. The work in [29] considers VN link reconfiguration in response to changes in traffic demands. That work does not allow for moving the placement of the virtual components. Recent work [30] has developed and implemented a scheme for moving a single virtual router in a virtualized separated router environment. This work has recently been used in [31] to design entire network migration mechanisms. A recent effort [32] has also considered this VN migration within software defined networks (SDN). All these recent works deal with low level migration configuration functions with no investigation of strategy and higher-level mechanism questions.

An alternative solution would be to dilate (spread) the critical footprint by dividing the traffic equally across all available routes. Major shortcoming of such approach will be a significant increase in the power consumption [33] and network management cost of the network. Also, it would require changing routing protocols as they automatically converges to a power-law (shortest path) based path logic after a failure.

III. AGILE VN FRAMEWORK

The objective of agile VN framework is to develop a migration strategy that determines what resources should be reassigned and, where and when migration should take place to defy hostile environment. In the following of the paper, we explain all the modules of agile VN framework, as in Figure 1.

A. Modeling VN Placement

The VN placement is all about finding the best combination of substrate (physical) resources satisfying all the requirements of one or more services, where each service will have its own VN. These requirements are enforced as constraints of the VN placement model, explained in the following. Let the substrate network be a directed graph, \( G = (V, E) \), where \( V \) is the set of all components (any network device) and \( E \) is...
where, \( c \) is true if component \( c \) and \( u \) are optimal paths between these source-destination pairs. Where \( C \) is the set of all source components in the network and \( C_{dst} \) is the set of all destination components. Both of such components represent a small fraction of the network size. Then, the VN placement is all about finding optimal paths between these source-destination pairs. Where \( u_i \) represents such a source-destination pair, \( u_i = < c_j, c_k > \) and \( u_i \in U = \{ C_{src} \times C_{dst} \} \). The shortest distance between any node \( c_j \) to a destination node \( c_k \) is represented as \( d_{ij} \), where, \( c_j \in C_{dst} \). And, \( b_{ik} \) is a Boolean variable, which is true if component \( c_k \) is carrying the traffic for a source-destination pair \( u_i \). \( I_{cj} \) is the set of neighboring components of \( c_j \) and \( \kappa_j \) is the traffic capacity of a node \( c_j \). Deploying a VN for a service requires the estimated traffic that needs to be handled by the network, e.g., between a corporate office to a data center\(^1\) and \( \lambda_i \) is the estimated traffic load of a pair \( u_i \). All thresholds in our model are represented as \( T^{(\lambda)} \).

The following placement model assigns substrate resources to multiple VNs. Let \( A \) be the set of all VNs, i.e., \( A = \{ v_1, v_2, \ldots, v_m \} \). Then, the binary assignment variable \( b_{ij} \) will be updated as, \( b_{ij}^{(\lambda)} \), which means, node \( c_j \) is assigned to pair \( u_i \) of VN \( v_j \). For the simplification of notations and understanding, we explain the VN placement modeling for one VN in the following section.

From the existing literature [6]–[10] we are using following requirements as constraints of our model.

1) Reachability Constraints: According to this constraint, data generated from a source must reach the destination. Assuming that each node will forward the traffic, this constraint is all about connecting components so that from source to destination there is a connected path. This constraint is formally defined as:

\[
 b_{ij} \rightarrow \bigvee_{\exists c_k \in I_{cj}} (b_{ij} \land b_{ik})
\]

which says, if a node is selected i.e., \( b_{ij} \) is true, then it must be connected to one of its neighboring components. In other words, a node receiving traffic must forward this to one of its neighboring component. But, forwarding a packet to one of the neighboring components does not guarantee packet delivery to the destination. So, in this constraint we need to add a sense of direction from source to destination. This sense can be added by using the distance from the current node to the destination node. The updated reachability constraint is defined in the following as:

\[
 b_{ij} \rightarrow \bigvee_{\exists c_k \in I_{cj}} ((b_{ij} \land b_{ik}) \land (d_{kl} < d_{jl}))
\]

where, \( d_{ij} \) is the distance from node \( c_j \) to destination node \( c_k \) of the pair \( u_i = < c_i, c_l > \). Now, what if the next node is itself the destination node, then traffic must not be forwarded to any other node. This completes the reachability constraint and the final version of this constraint is as follows:

\[
 b_{ij} \rightarrow (c_j \in C_{dst}) \lor \bigvee_{\exists c_k \in I_{cj}} ((b_{ij} \land b_{ik}) \land (d_{kl} < d_{jl})) \quad (1)
\]

Although, we assume mostly stable routing because according to [1] 72% of links tend to be stable. However, our model can be incrementally updated to recalculate placement in case of topology changes.

2) Load Satisfaction Constraint: According to this constraint, every node along the path must be able to carry traffic load of the pair assigned to it. In practical, this decision must be made by comparing traffic load with leftover capacity of the node. If the aggregate traffic supported by node \( c_j \) is \( \sigma_j \), which is calculated in the Section III-A5, then formal definition of this constraint is as followed:

\[
 \forall u_i \in U, c_j \in C, b_{ij} \rightarrow (\kappa_j - \sigma_j) \geq \lambda_i \quad (2)
\]

3) Middle-ware Device Constraint: An application might require some specialized service from the substrate (e.g., the IPSec, etc.). Mostly, only a handful of components can provide such requested feature. This constraint ensures that the VN traffic must go through such specialized node. Let \( C_{sp} \subset C \) be a set of such specialized components. Then, for a pair \( u_i \) we enforce this constraint by explicitly setting the node assignment variable \( b_{ij} \) to true, where, \( c_j \in C_{sp} \). The formal definition is as followed:

\[
 \exists c_j \in C_{sp}, b_{ij} \quad (3)
\]

The VN placement model, as a whole, will automatically connect this component with other components to satisfy all constraints.

4) Quality of Service Constraint: For simplicity, we assume relatively homogeneous network where hop count can be proportional to network latency. This assumption has been widely used in existing literature [34].

\[
 \forall u_i \in U, (\sum_{c_j \in C} b_{ij}) \leq T^{qs} \quad (4)
\]

where, \( T^{qs} \) is the affordable delay threshold for any pair in the system. It can be the same for the entire system or different for each VN. In addition, our framework allows for more complex constraints to define bounded delays.

\(^1\)In a commercial products like Virtela [11] and Aryaka [12], they expect to estimate this traffic as well.
5) **Node Stress Constraint:** According to this constraint, each node must have some leftover capacity to accommodate the traffic fluctuations due to load dynamics or uncertainty. Therefore, the leftover capacity can ensure to handle such incorrect traffic load estimations. This is formally defined as:

\[ \forall c_j \in C, \{ (\kappa_j - (\sum_{u_i \in U} b_{ij} \lambda_i)) \geq T^{lo} \} \tag{5} \]

where, \( \sum_{u_i \in U} b_{ij} \lambda_i \) is the aggregate traffic \( \sigma_j \) at node \( c_j \) and \( T^{lo} \) is the leftover capacity threshold set for each component. This can be provided as a percentage of the capacity of a node.

6) **Pair Mapping Constraint:** According to this constraint, each pair must be assigned some substrate resources. This is formally defined as:

\[ \forall u_i \in U, \sum_{c_j \in C} b_{ij} > 1 \tag{6} \]

7) **Loop Avoidance Constraint:** According to this constraint, a substrate component cannot be assigned to a pair twice because this would create a loop within the path. Our model enforces this by checking how many links are enabled for each component? In the model, a link is represented logically as \( b_{ij} \land b_{ik} \), i.e., both components \( c_j \) and \( c_k \) are assigned to \( u_i \) and they are connected. If the number of links are more than 2, this means there is a loop. E.g., \( c_j \land c_k \) and \( c_k \land c_i \), if \( c_j \land c_j \), this makes a loop that is not allowed. It is formally defined as follow:

\[ \forall u_i \in U, c_j \in C, (\sum_{c_k \in I_{c_j}} (b_{ij} \land b_{ik} \rightarrow 1)) \leq 2 \tag{7} \]

In this equation the condition is not modeled as exactly equal to 2 but less than or equal to 2. It is because, the source and destination components will have only one link.

IV. MODELING THREAT

The planning phase of all link-level DDoS attacks is to perform network reconnaissance to identify critical components. For proactive defense, we have to model this reconnaissance attack and incorporate it as a module of the agile VN framework. This module proactively identifies potential targets so that these can be replaced with threat safe components to evade attacks. Our framework can handle general classes of link-level DDoS and reconnaissance attacks. It is general enough to consider new attack by only updating threat model to reflect new logic of identifying critical components without changing any other parts of the framework in Figure 1. However, in this paper we are using the most devastating and stealthy DDoS attack to-date, i.e., Crossfire [1] as a case study threat model.

In the Internet, data connectivity follows power law distribution [4], which means a small set of links carry most of the traffic. The Crossfire attack identifies such links that carry most of the traffic to the victim. Then, it selects a list of surrounding public servers (decoy servers) that share these links with the target. The attack uses botnets to send enough traffic to these selected decoy servers that it throttles the shared links, thus causing denial of service to the victim, indirectly. The static critical status quo of these links enables the adversary to launch such attack.

Under the hood of a Crossfire attack, bots send traceroute probes to all victim machines and decoy servers. As opposed to manually searching decoy servers in the original attack, we use the geographical coordinates of the victim and select initial set of decoy servers who are within certain miles from the victim. We use geographical database GeoLite [35] to get all such coordinates and use geographical distance formula [36] to calculate distance. The final set of decoy servers is selected after observing the sharing of infrastructure between victim and decoy servers. Let the set of bots be \( O \), decoy servers be \( Y \) and victim machines be \( V \). Each traceroute probe provides a data path, and \( P \) is the set of all paths from bots to decoy servers and victim machines. And, \( P^Y \) and \( P^V \) are the set of paths from bots to decoy servers and victim machines, respectively, i.e., \( P = \{ P^Y \cup P^V \} \). A path is simply a set of components, i.e., \( p = \{c_1, c_2, \ldots, c_n\} \), where, \( c_n \in C \) is from Section III-A. All those paths that do not share anything between victim and decoy servers are removed from the set, i.e., \( P' = \{\neg \exists p_i \in P^Y, p_j \in P^V, p_i \cap p_j = \emptyset\} \).

For a packet to be delivered successfully from origin to destination, every component along the path must forward the packet. Let \( F(.) \) be a Boolean function, which is true if the component successfully forwards the packet and false otherwise. This path forwarding property can be expressed as a logical formula as, \( F(p_i) = \bigwedge_{c_j \in p_i} F(c_j) \). However, from the adversary’s perspective, she just has to compromise any single component along the path to violate the forwarding property of the path. Let \( B(.) \) be a Boolean function that is true if a component along the path is compromised and false otherwise. Now, the same path forwarding property can be flipped to formulate path breaking property, which says a path is broken if any component along the path is compromised, i.e., \( B(p_i) = \bigvee_{c_j \in p_i} B(c_j) \). Due to limited resources of the adversary, compromising the entire network may not be a possibility. Therefore, the model must also be equipped with a choice to disrupt any path or not based on the budget of the adversary. This choice is added as a binary variable \( \chi \) in the path breaking property, i.e., \( B(p_i) = \bigvee_{c_j \in p_i} B(c_j) \rightarrow \chi \). The value of this choice variable, \( \chi \), will be one if the path can be broken or zero otherwise. This all depends upon the budget of the adversary. We can extend this path breaking property for the entire network to find a set of components, if broken, can cause the desired devastation in the network. Its formal
This formula provides a list of components, if attacked, can inflict the desired devastation. This is an NP complete problem and we formalize it as a SAT [13] problem. Normally, the adversary has limited budget (cost) in terms of attack traffic that can be generated to throttle components. Therefore, finding only those components that can be throttled with that budget is the prime focus of adversary. Let \( |B(\cdot)| \) be a function that provides number of true components in Equation 8. If the adversary does not know the capacity of target components, then cost will represent the number of components that can be throttled assuming them to have same capacity. This constraint is encoded as follows:

\[
|B(P^t)| \leq \text{Cost} \quad (9)
\]

\[
\sum_{\forall p_i \in P^t} \chi_i \leq T^{dv} \quad (10)
\]

where \( T^{dv} \) is the desired disruption of the adversary. If we assume that the adversary has the leftover capacity information of each component then the Equation 9 can be optimized. This last assumption is optional and we do not use this in our implementation.

\[
\sum_{\forall c_j \in C} ((\kappa_j - \sigma_j) \ast B(c_j)) \leq \text{Cost} \quad (11)
\]

During reconnaissance attack, the adversary cannot afford to send too much traffic to decoy servers or victim machines to avoid detection. Therefore, by simply adding more bots will not guarantee that the reconnaissance attack will complete in small time window. The experiments explained in Section VII-B5 provide the reconnaissance time benchmark of this attack model.

A. Defender’s View vs Attacker’s View

When an adversary runs a traceroute probe for a target machine, she views the same data plane that is deployed during the VN placement for the service running on that target machine. Similar observation is made in [1] that bots probing a target from all over the world would end up viewing the same critical footprint with respect to the target. Therefore, the threat model discussed in this section can be viewed as both defender’s and attacker’s perspective of potential targets. Because, the decoy server information is public and available to both. Similarly, the adversary would rely on bot to decoy server relation and defender would rely on source to decoy server relation to identify critical components. Where bots and sources are logically the same just with different roles. In Section VII-B6, we have demonstrated this fact using simulation to show that both see the same components as potential targets.
implementation on PlanetLab, we use controller based architecture [17] like in Software Defined Networking (SDN). In such architecture, the relative positions of the current and new components do not matter. Instead, control plane resides at the controller, which populates the forwarding plane in the new components by sending control packets carrying forwarding rules. The controller already has established tunnels with each network component. So, the migration time will be calculated as: how much time will it take to send control packets from the controller to the new substrate components. Let $c_j$ be the distance from the controller to a component $c_j$, $\tau$ be the time spent per hop and $b_{ij}^t$, where $t$ represents the index of the iteration of migration. Now, we can define our migration distance constraint as:

$$\left\{ \sum_{u_i \in U} \sum_{c_j, c_k \in C} ((b_{ij}^t \land b_{jk}^{t+1}) \land (j \neq k)) \rightarrow \tau \ast \alpha_k \right\} < \omega \quad (13)$$

where, $b_{ij}^{t+1}$ represents the assignment of component $c_j$ to pair $u_i$ in the new placement. To quantify this equation, we use calculations from our PlanetLab experiments. In our PlanetLab experiments, we observed the average traceroute probe time to be almost 250ms and the average path length observed is almost 10 hops. We use these measures to develop following heuristics. The traceroute time represents a round trip time, so, one way time will be 125ms. This means it takes almost 13ms per hop along the path. Assuming the controller can initiate populating forwarding tables at once for all nodes, then using a more conservative hop time i.e., $\tau = 20ms$ we can calculate the migration distance in Equation 13. We assume that the size of forwarding tables is the same and bandwidth from controller to node will remain the same. Therefore, we do not need to put these as the part of the model as they are not going to change for different controller-node pairs.

VI. MIGRATION MECHANISM

The migration mechanism executes VN migration strategy generated by the framework on a virtualized physical network. Given the destination for a particular VN to be moved, it handles all the logistics of the movement including the exact sequence/schedule of the steps to be executed in order to complete the move and the timing of such steps. It is also responsible for interacting with specific substrate and its virtualization technology to accomplish the move. There are two main parts of the mechanism (in Figure 2), (1) the scheduler is responsible to determine exact sequence of moves based on the input migration strategy from the agile VN framework while the configuration manager implements and executes these steps on the infrastructure like PlanetLab [16]. It reserves resources needed during the migration, sending signals to virtual nodes ordering them to start their moves, keeping track of the health of the migration via monitoring, and requesting dynamic adjustments to the migration schedule if necessary. We have used PlanetLab as our testbed to implement migration mechanism because it offers a unique geographically distributed virtualized infrastructure. Instead of developing the entire mechanism from scratch, we leverage an existing PlanetLab based controller, PL-VNM [17]. This controller offers the core migration functionality and implements single virtual node migration. An abstract level view of our extensions to PL-VNM are briefly described as follows.

A. Implementing VN Placement

We have implemented our VN placement module inside the controller. Now, the controller can perform two key tasks, (1) given a substrate topology inside the PlanetLab, the controller can find the initial suitable placement, and (2) using the threat model it can also generate a migration strategy to initiate a threat aware migration.

B. Implementing Threat Model

We have also implemented the Crossfire threat model as the part of the controller. For a complete threat simulation, we also designate different PlanetLab nodes (allocated to us) as bots and decoy servers manually. As each node might be located in a different country, there is no way of automating the decoy server selection using geographical coordinates. The threat module instructs these bots to send traceroute probes to each decoy server and then collects the path information to find critical footprint. Later, the bots are instructed to start sending attack traffic using already configured UDP based iperf.

C. Implementing Partial Migration

The controller was lacking the partial VN migration capability. In the existing implementation, all end users connect to specialized nodes that are called gateways. These gateways were responsible to switch from old substrate to new substrate. This method is used to migrate entire VN from one substrate to another. Our framework requires each node to act as a gateway, as any node can be flagged as critical. Once a node is identified as critical, all of its neighboring nodes must act as gateways to move traffic away from this critical node. We have extended the role of each node to act as a gateway in the controller.

VII. IMPLEMENTATION AND EVALUATION

We have used PlanetLab, simulation and Mininet based experiments to perform a rigorous evaluation.

A. Experiment Setup Discussion

1) PlanetLab based Experiment Setup: The experiment topology that we have developed within the PlanetLab is showed in Figure 3. In our experiments, we classify the components into four categories: 1) fixed components, the components that do not change during migration, black ones in the figure; 2) Gateway components, that switch traffic between new and previous components, respectively. These are the yellow components $c_b$ and $c_{17}$ in the figure; 3) Critical components, that are needed to be changed, these are $c_9$, $c_8$ and $c_{10}$ in migration step 1, 2 and 3, respectively; 4) Origin and sink components in the network, e.g., $c_1$, $c_{18}$ and $y_1$ are

3Due to limited space, we skip the technical details of our controller extensions and implementation.
the source, the destination and the decoy server components, respectively.

The same color of bots, components and links in Figure 3 represents that these were active during the same migration iteration. For example, during iteration 1, the component $c_9$ was selected as critical (only one carrying traffic to the target) and only bots $o_1$ through $o_4$ were activated in the attack because these are the only bots sharing critical component $c_9$ with the destination. Similarly, the path from source $c_1$ to destination $c_{18}$ was {$c_3, c_6, c_9, c_{17}$} and as the links between $c_6$ to $c_9$ and from $c_9$ to $c_{17}$ were carrying the traffic during iteration 1, they all have the same color. Another pivotal concept observable in this example is that the changing of the critical footprint also changes the association between bots and decoy servers. This means that the adversary has to repeat reconnaissance to learn this new association again.

![Available Bandwidth Analysis](a)

![Migration Time](b)

![Packet Loss during Migration](c)

![Bandwidth Analysis in PlanetLab](d)

![Bandwidth Analysis in Mininet](e)

![Critical Footprint Size](f)

![Critical Footprint Size (%age)](g)

![Reconnaissance Time](h)

![Attacker’s vs. Defender’s View](i)

Fig. 4. These diagrams present the effectiveness and disruptiveness of agility results.

![Depth of Defense](a)

![Time w.r.t. Network Size](b)

![Time w.r.t. Multiple VNs](c)

Fig. 5. This diagram shows Migration overhead, Framework scalability results.

<table>
<thead>
<tr>
<th>Network Size</th>
<th>Preferential Network</th>
<th>Random Network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of Rules</td>
<td>No. of MBs</td>
</tr>
<tr>
<td>100</td>
<td>600000</td>
<td>145.2</td>
</tr>
<tr>
<td>150</td>
<td>900000</td>
<td>217.8</td>
</tr>
<tr>
<td>200</td>
<td>1200000</td>
<td>293.4</td>
</tr>
<tr>
<td>600</td>
<td>3600000</td>
<td>871.2</td>
</tr>
</tbody>
</table>

2) Simulation based Experiment Setup: We have used BRITE [37] to generate topologies with random connectivity and power law based preferential connectivity. In each topology, a small fraction of nodes (5%) with least degree are designated as sources and destinations. The shortest paths between these nodes simulate network data plane. First, VN placement module finds different placement, second, threat model finds critical footprint in each placement. We randomly select least
degree nodes as bots and near by nodes to destination (based on hop distance) as decoy server for the threat model. We use Z3 theorem prover to formulate a constraints satisfaction model [14]. The Z3 is a state-of-the-art theorem prover from Microsoft Research and it is used to check the satisfiability of logical formulas over one or more theories. We have used a machine with Core i7, 2.4 GHz processor and 8GB RAM to run all the simulation based experiments.

3) Mininet based Experiment Setup: Mininet based experiments are used to observe packet loss during the migration. We have used POX as a reference controller for Mininet. We create similar topology as in Figure 3, but without any bots and decoy servers. We have extended the POX controller by adding a switching module for our implementation. We have used the iperf functionality built-in the Mininet to observe packet loss during switching.

B. Agile VN Framework Evaluation

1) Evaluating Depth of Defense: The depth of defense measures the over provisioning available in terms of alternate routes and placements available in the Internet. Alternate routes provisioning is calculated using PlanetLab experiments and alternate VN placements is calculated using simulation. In simulation experiments, we let the model calculates all possible placements for each network type and size, and results are showed in Figure 5(a). The results demonstrate that in every network, we always have a decent number of alternate VN placements available, e.g., almost 100 for a network of size 700 nodes, and that increase with the increase in the size of the network.

To calculate the depth of defense in terms of alternate routes, we selected three PlanetLab university sites, from East Coast of USA, West Coast of USA and Europe. Then, we select 100 PlanetLab nodes scattered through the world to send traceroute probes to these university sites. Each probing node sent exactly 6 probes (following same model as in [1]). After extracting the paths from these probes, we calculate alternate routes by simply identifying the minimum number of links that need to be broken to compromise all paths. The results showed in Table II demonstrate that there are always 4 to 5 alternate paths available to each destination. Now, as commercial overlay virtualization providers [11], [12] are in the picture, exhibiting central authority, this over provision will be more because in probing we do not necessarily view all links e.g., backup links.

<table>
<thead>
<tr>
<th>East Coast</th>
<th>West Coast</th>
<th>Europe</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

2) Evaluating the Evasion Effectiveness: In evaluating evasion effectiveness experiment on topology in Figure 3, we calculated link bandwidth between source (c1) and destination (c15) nodes using TCP based Iperf. It was observed sometimes as 2.0 Mbps and sometimes 1.6 Mbps due to bandwidth limit enforced by PlanetLab, results are showed in Figure 4(a). At around 13th second, the Crossfire attack was launched from bots o4 through o4 that crippled the available bandwidth by limiting it to 300 Kbps within seconds. At time 27th, the agility module of the PL-VNM is activated to initiate evasion through migrating to a different path and that instantly restored the bandwidth back to 2.0 Mbps. We let this experiment run for sometimes to calculate the migration time in switching between nodes. The results in Figure 4(b) show that the migration time stayed around 1 to 2 seconds which is way better than the reconnaissance time observed in Figure 4(h). During these experiments we kept the SSH channel open between the controller and the nodes.

3) Evaluating the Disruptiveness of Migration: We design an experiment on PlanetLab using the same topology from Figure 3 to analyze the packet loss caused by the migration. Without any attack, we let the controller keep migrating between nodes in every 2 minutes and we calculated the average bandwidth available in each minute. The cumulative distribution function (CDF) results in Figure 4(c) show only 5% packet loss for almost 85% of the times and higher for just 15% of the times. Further investigation revealed that PlanetLab uses unicast reverse path forwarding. It matches the arriving interface of the packet with the departure interface, if it does not match, it simply drops packets. That is why all in-flight packets were lost every time the migration happens. This is the limitation with PlanetLab architecture and there is nothing that can be done. Therefore, we tested the same migration experiment in a more migration friendly virtualized infrastructure, i.e., Mininet. The combined results in PlanetLab and Mininet experiments, under all possible scenarios, are presented in Figure 4(d) and Figure 4(e), respectively. And the Mininet results in Figure 4(e) clearly show that there is almost no or minimal packet loss observed during migration. This proves that if the network is supportive of the migration then packet loss will not be an issue.

4) Evaluating the Overhead of Migration: We have used simulation to calculate migration overhead in terms of nodes to be migrated and extra traffic to be generated because, on PlanetLab such large scale experiments were not possible. The Figures 4(f) and 4(g) demonstrate the overhead results in terms of the size of critical footprint and its percentage w.r.t. the network size respectively, for different networks. We have used both types of networks (preferential and random) to evaluate the critical footprint size. For networks with preferential connectivity, the size of critical profile represents around 5 – 10% of the network, whereas, for random network this percentage is 9 – 14%. Intuitively, preferential connectivity based networks have a smaller critical footprint than random ones.

The amount of overhead traffic is calculated by multiplying average routing table size, which is 24.4 MB or 10K rules [30], with the number of components to be migrated. Results with different network sizes are showed in Table I. For a network of 100 nodes, in both preferential and random networks, overhead traffic size is only 145 MB and 217

4Due to anonymous submission, we are not disclosing this information.
MB respectively. And for a large network of the size 600 nodes, it is 871 MB and 1.8 GB for preferential and random networks respectively. In large networks, this not significant traffic because they have to over provision bandwidth to handle DDoS amounting to hundreds of giga bits e.g., in 2014, CloudFair’s customer was hit with a massive 400 Gbps DDoS attack [38]. Furthermore, reconnaissance in large networks also takes more time and this helps to reduce the frequency of migration.

5) Benchmarks of Reconnaissance Time: The VN agility is bounded by reconnaissance time, which we calculate through a PlanetLab experiment. In this experiment, we host a web server on one of the USA based university node of PlanetLab and select 10 to 70 decoy server around that university. We use only one PlanetLab node as bot to perform reconnaissance because, in actual Crossfire reconnaissance, each node has to perform same amount of probing. Furthermore, increasing bots will not increase time because of limited probing traffic that can be sent to decoy servers and targets to avoid detection. The results in Figure 4(h) show that even for a small scaled experiment with just 70 decoys, it took us 5 minutes to complete reconnaissance, that provide a big margin to maneuver.

6) Evaluating Attacker’s View vs. Defender’s View: We use simulation environment, explained in Section VII-A2, to calculate disparity, if any, between the defender’s view and the attacker’s view of the critical footprint. They both view the same network data plane and decoy server information is publicly available. We start by finding critical footprint sets, firstly, by just using bots (for attacker) and secondly, by just using sources (for defender). Then, we calculate overlapping ratio of two sets. The results in Figure 4(i) clearly show that almost 95% of the time both end up selecting the same critical footprint that coincide the finding of the existing literature [1].

7) Evaluating Scalability: We tested the scalability of our approach in terms of time complexity by varying the size of the networks and number of VNs within a network. The results of varying network size and number of VNs are showed in Figure 5(b) and Figure 5(c) respectively. The time complexity increases linearly with network size and for network of the size 700 components, it is well below one minute. Whereas, it does not change much by just increasing the number of VNs within same network e.g., it changes from 33 seconds to 38 seconds for one VN to 50 VNs in a network of size of 500.

VIII. CONCLUSION

All DDoS attacks focus only on a small set of critical network components. If we can change the role of these critical components to non-critical ones, we can evade DDoS attacks. Virtual networks provide this flexibility by dynamically assigning and reassigning physical resources to a service. In this paper, we have proposed a correct-by-construction agile VN framework that proactively defended against sophisticated DDoS attacks like Crossfire by actively reassigning the VN to new threat safe physical resources and without breaking the service or violating the VN properties. We have implemented that framework on PlanetLab and our experiments showed the effectiveness of restoring the downgraded bandwidth (80%) due to DDoS attack by migrating to a threat safe placement in just seconds.

REFERENCES


