Bohatei: Flexible and Elastic DDoS Defense

Seyed K. Fayaz, Yoshiaki Tobioka, and Vyas Sekar, *Carnegie Mellon University;*
Michael Bailey, *University of Illinois at Urbana-Champaign*

https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/fayaz

This paper is included in the Proceedings of the
24th USENIX Security Symposium

August 12–14, 2015 • Washington, D.C.

ISBN 978-1-931971-232

Bohatei: Flexible and Elastic DDoS Defense

Seyed K. Fayaz[*] *Yoshiaki Tobioka*[*] *Vyas Sekar*[*] *Michael Bailey*[†]
[*]*CMU* [†]*UIUC*

Abstract

DDoS defense today relies on expensive and proprietary hardware appliances deployed at fixed locations. This introduces key limitations with respect to flexibility (e.g., complex routing to get traffic to these "chokepoints") and elasticity in handling changing attack patterns. We observe an opportunity to address these limitations using new networking paradigms such as software-defined networking (SDN) and network functions virtualization (NFV). Based on this observation, we design and implement Bohatei, a *flexible* and *elastic* DDoS defense system. In designing Bohatei, we address key challenges with respect to scalability, responsiveness, and adversary-resilience. We have implemented defenses for several DDoS attacks using Bohatei. Our evaluations show that Bohatei is scalable (handling 500 Gbps attacks), responsive (mitigating attacks within one minute), and resilient to dynamic adversaries.

1 Introduction

In spite of extensive industrial and academic efforts (e.g., [3, 41, 42]), distributed denial-of-service (DDoS) attacks continue to plague the Internet. Over the last few years, we have observed a dramatic escalation in the number, scale, and diversity of DDoS attacks. For instance, recent estimates suggest that over 20,000 DDoS attacks occur per day [44], with peak volumes of 0.5 Tbps [14, 30]. At the same time, new vectors [37, 55] and variations of known attacks [49] are constantly emerging. The damage that these DDoS attacks cause to organizations is well-known and include both monetary losses (e.g., $40,000 per hour [12]) and loss of customer trust.

DDoS defense today is implemented using expensive and proprietary hardware appliances (deployed in-house or in the cloud [8, 19]) that are *fixed* in terms of placement, functionality, and capacity. First, they are typically deployed at fixed network aggregation points (e.g., a peering edge link of an ISP). Second, they provide fixed functionality with respect to the types of DDoS attacks they can handle. Third, they have a fixed capacity with respect to the maximum volume of traffic they can process. This fixed nature of today's approach leaves network operators with two unpleasant options: (1) to overprovision by deploying defense appliances that can handle a high (but pre-defined) volume of every known attack type at each of the aggregation points, or (2) to deploy a smaller number of defense appliances at a central location (e.g., a scrubbing center) and reroute traffic to this location. While option (2) might be more cost-effective, it raises two other challenges. First, operators run the risk of underprovisioning. Second, traffic needs to be explicitly routed through a fixed central location, which introduces additional traffic latency and requires complex routing hacks (e.g., [57]). Either way, handling larger volumes or new types of attacks typically mandates purchasing and deploying new hardware appliances.

Ideally, a DDoS defense architecture should provide the *flexibility* to seamlessly place defense mechanisms where they are needed and the *elasticity* to launch defenses as needed depending on the type and scale of the attack. We observe that similar problems in other areas of network management have been tackled by taking advantage of two new paradigms: software-defined networking (SDN) [32, 40] and network functions virtualization (NFV) [43]. SDN simplifies routing by decoupling the control plane (i.e., routing policy) from the data plane (i.e., switches). In parallel, the use of virtualized network functions via NFV reduces cost and enables elastic scaling and reduced time-to-deploy akin to cloud computing [43]. These potential benefits have led major industry players (e.g., Verizon, AT&T) to embrace SDN and NFV [4, 6, 15, 23].[1]

In this paper, we present Bohatei[2], a flexible and

[1] To quote the SEVP of AT&T: "To say that we are both feet in [on SDN] would be an understatement. We are literally all in [4]."

[2] It means breakwater in Japanese, used to defend against tsunamis.

elastic DDoS defense system that demonstrates the benefits of these new network management paradigms in the context of DDoS defense. Bohatei leverages NFV capabilities to elastically vary the required scale (e.g., 10 Gbps vs. 100 Gbps attacks) and type (e.g., SYN proxy vs. DNS reflector defense) of DDoS defense realized by defense virtual machines (VMs). Using the flexibility of SDN, Bohatei steers suspicious traffic through the defense VMs while minimizing user-perceived latency and network congestion.

In designing Bohatei, we address three key algorithmic and system design challenges. First, the resource management problem to determine the number and location of defense VMs is NP-hard and takes hours to solve. Second, existing SDN solutions are fundamentally unsuitable for DDoS defense (and even introduce new attack avenues) because they rely on a per-flow orchestration paradigm, where switches need to contact a network controller each time they receive a new flow. Finally, an intelligent DDoS adversary can attempt to evade an elastic defense, or alternatively induce provisioning inefficiencies by dynamically changing attack patterns.

We have implemented a Bohatei controller using `OpenDaylight` [17], an industry-grade SDN platform. We have used a combination of open source tools (e.g., OpenvSwitch [16], Snort [48], Bro [46], iptables [13]) as defense modules. We have developed a scalable resource management algorithm. Our evaluation, performed on a real testbed as well as using simulations, shows that Bohatei effectively defends against several different DDoS attack types, scales to scenarios involving 500 Gbps attacks and ISPs with about 200 backbone routers, and can effectively cope with dynamic adversaries.

Contributions and roadmap: In summary, this paper makes the following contributions:

- Identifying new opportunities via SDN/NFV to improve the current DDoS defense practice (§2);
- Highlighting the challenges of applying existing SDN/NFV techniques in the context of DDoS defense(§3);
- Designing a responsive resource management algorithm that is 4-5 orders of magnitude faster than the state-of-the-art solvers (§4);
- Engineering a practical and scalable network orchestration mechanism using proactive tag-based forwarding that avoids the pitfalls of existing SDN solutions (§5);
- An adaptation strategy to handle dynamic adversaries that can change the DDoS attack mix over time (§6);
- A proof-of-concept implementation to handle several known DDoS attack types using industry-grade SDN/NFV platforms (§7); and
- A systematic demonstration of the scalability and effectiveness of Bohatei (§8).

We discuss related work (§9) before concluding (§10).

2 Background and Motivation

In this section, we give a brief overview of software-defined networking (SDN) and network functions virtualization (NFV) and discuss new opportunities these can enable in the context of DDoS defense.

2.1 New network management trends

Software-defined networking (SDN): Traditionally, network control tasks (e.g., routing, traffic engineering, and access control) have been tightly coupled with their data plane implementations (e.g., distributed routing protocols, ad hoc ACLs). This practice has made network management complex, brittle, and error-prone [32]. SDN simplifies network management by decoupling the network control plane (e.g., an intended routing policy) from the network data plane (e.g., packet forwarding by individual switches). Using SDN, a network operator can centrally program the network behavior through APIs such as OpenFlow [40]. This flexibility has motivated several real world deployments to transition to SDN-based architectures (e.g., [34]).

Network functions virtualization (NFV): Today, network functions (e.g., firewalls, IDSes) are implemented using specialized hardware. While this practice was necessary for performance reasons, it leads to high cost and inflexibility. These limitations have motivated the use of virtual network functions (e.g., a virtual firewall) on general-purpose servers [43]. Similar to traditional virtualization, NFV reduces costs and enables new opportunities (e.g., elastic scaling). Indeed, leading vendors already offer virtual appliance products (e.g., [24]). Given these benefits, major ISPs have deployed (or are planning to deploy) datacenters to run virtualized functions that replace existing specialized hardware [6, 15, 23]. One potential concern with NFV is low packet processing performance. Fortunately, several recent advances enable line-rate (e.g., 10-40Gbps) packet processing by software running on commodity hardware [47]. Thus, such performance concerns are increasingly a non-issue and will further diminish given constantly improving hardware support [18].

2.2 New opportunities in DDoS defense

Next, we briefly highlight new opportunities that SDN and NFV can enable for DDoS defense.

Lower capital costs: Current DDoS defense is based on specialized hardware appliances (e.g., [3, 20]). Network operators either deploy them on-premises, or outsource DDoS defense to a remote packet scrubbing site (e.g., [8]). In either case, DDoS defense is expensive.

For instance, based on public estimates from the General Services Administration (GSA) Schedule, a 10 Gbps DDoS defense appliance costs ≈\$128,000 [11]. To put this in context, a commodity server with a 10 Gbps Network Interface Card (NIC) costs about \$3,000 [10]. This suggests roughly 1-2 orders of magnitude potential reduction in capital expenses (ignoring software and development costs) by moving from specialized appliances to commodity hardware.[3]

Time to market: As new and larger attacks emerge, enterprises today need to frequently purchase more capable hardware appliances and integrate them into the network infrastructure. This is an expensive and tedious process [43]. In contrast, launching a VM customized for a new type of attack, or launching more VMs to handle larger-scale attacks, is trivial using SDN and NFV.

Elasticity with respect to attack volume: Today, DDoS defense appliances deployed at network chokepoints need to be provisioned to handle a predefined maximum attack volume. As an illustrative example, consider an enterprise network where a DDoS scrubber appliance is deployed at each ingress point. Suppose the projected resource footprint (i.e., defense resource usage over time) to defend against a SYN flood attack at times t_1, t_2, and t_3 is 40, 80, and 10 Gbps, respectively.[4] The total resource footprint over this entire time period is $3 \times max\{40,80,10\} = 240$ Gbps, as we need to provision for the worst case. However, if we could elastically scale the defense capacity, we would only introduce a resource footprint of $40+80+10 = 130$ Gbps—a 45% reduction in defense resource footprint. This reduced hardware footprint can yield energy savings and allow ISPs to repurpose the hardware for other services.

Flexibility with respect to attack types: Building on the above example, suppose in addition to the SYN flood attack, the projected resource footprint for a DNS amplification attack in time intervals t_1, t_2, and t_3 is 20, 40, and 80 Gbps, respectively. Launching only the required types of defense VMs as opposed to using monolithic appliances (which handle both attacks), drops the hardware footprint by 40%; i.e., from $3 \times (max\{40,80,10\} + max\{20,40,80\}) = 480$ to 270.

Flexibility with respect to vendors: Today, network operators are locked-in to the defense capabilities offered by specific vendors. In contrast, with SDN and NFV, they can launch appropriate best-of-breed defenses. For example, suppose vendor 1 is better for SYN flood defense, but vendor 2 is better for DNS flood defense. The physical constraints today may force an ISP to pick only one hardware appliance. With SDN/NFV we can avoid the undesirable situation of picking only one vendor and rather have a deployment with both types of VMs each for a certain type of attack. Looking even further, we also envision that network operators can mix and match capabilities from different vendors; e.g., if vendor 1 has better detection capabilities but vendor 2's blocking algorithm is more effective, then we can flexibly combine these two to create a more powerful defense platform.

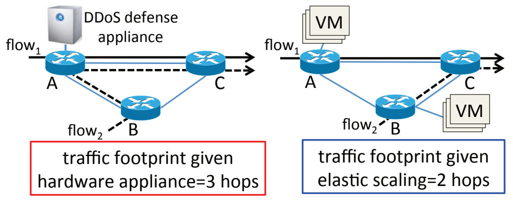

DDoS defense appliance
flow$_1$
A
C
flow$_2$
B
traffic footprint given hardware appliance=3 hops
VM
flow$_1$
A
C
flow$_2$
B
VM
traffic footprint given elastic scaling=2 hops

Figure 1: DDoS defense routing efficiency enabled by SDN and NFV.

Simplified and efficient routing: Network operators today need to employ complex routing hacks (e.g., [57]) to steer traffic through a fixed-location DDoS hardware appliance (deployed either on-premises or in a remote site). As Figure 1 illustrates, this causes additional latency. Consider two end-to-end flows $flow_1$ and $flow_2$. Way-pointing $flow_2$ through the appliance (the left hand side of the figure) makes the total path lengths 3 hops. But if we could launch VMs where they are needed (the right hand side of the figure), we could drop the total path lengths to 2 hops—a 33% decrease in traffic footprint. Using NFV we can launch defense VMs on the closest location to where they are currently needed, and using SDN we can flexibly route traffic through them.

In summary, we observe new opportunities to build a flexible and elastic DDoS defense mechanism via SDN/NFV. In the next section, we highlight the challenges in realizing these benefits.

3 System Overview

In this section, we envision the deployment model and workflow of Bohatei, highlight the challenges in realizing our vision, and outline our key ideas to address these challenges.

3.1 Problem scope

Deployment scenario: For concreteness, we focus on an ISP-centric deployment model, where an ISP offers DDoS-defense-as-a-service to its customers. Note that several ISPs already have such commercial offerings (e.g., [5]). We envision different monetization avenues. For example, an ISP can offer a value-added security service to its customers that can replace the customers' in-house DDoS defense hardware. Alternatively, the ISP can allow its customers to use Bohatei as a cloudbursting option when the attack exceeds the customers' on-

[3] Operational expenses are harder to compare due to the lack of publicly available data.

[4] For brevity, we use the traffic volume as a proxy for the memory consumption and CPU cycles required to handle the traffic.

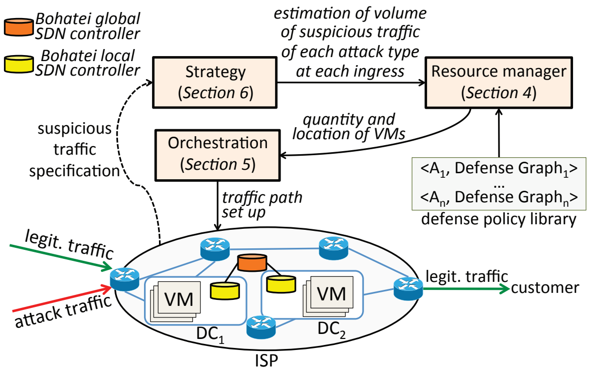

Bohatei global SDN controller
Bohatei local SDN controller
estimation of volume of suspicious traffic of each attack type at each ingress
Strategy (Section 6)
Resource manager (Section 4)
quantity and location of VMs
suspicious traffic specification
Orchestration (Section 5)
<A1, Defense Graph1>
...
<An, Defense Graphn>
defense policy library
traffic path set up
legit. traffic
attack traffic
VM
VM
DC1
DC2
legit. traffic
customer
ISP

Figure 2: Bohatei system overview and workflow.

premise hardware. While we describe our work in an ISP setting, our ideas are general and can be applied to other deployment models; e.g., CDN-based DDoS defense or deployments inside cloud providers [19].

In addition to traditional backbone routers and interconnecting links, we envision the ISP has deployed multiple datacenters as shown in Figure 2. Note that this is not a new requirement; ISPs already have several in-network datacenters and are planning additional rollouts in the near future [15,23]. Each datacenter has commodity hardware servers and can run standard virtualized network functions [45].

Threat model: We focus on a general DDoS threat against the victim, who is a customer of the ISP. The adversary's aim is to exhaust the network bandwidth of the victim. The adversary can flexibly choose from a *set of candidate attacks* $AttackSet = \{A_a\}_a$. As a concrete starting point, we consider the following types of DDoS attacks: TCP SYN flood, UDP flood, DNS amplification, and elephant flow. We assume the adversary controls a large number of bots, but the total *budget* in terms of the maximum volume of attack traffic it can launch at any given time is fixed. Given the budget, the adversary has a complete control over the choice of (1) type and mix of attacks from the *AttackSet* (e.g., 60% SYN and 40% DNS) and (2) the set of ISP ingress locations at which the attack traffic enters the ISP. For instance, a simple adversary may launch a single fixed attack A_a arriving at a single ingress, while an advanced adversary may choose a mix of various attack types and multiple ingresses. For clarity, we restrict our presentation to focus on a single customer noting that it is straightforward to extend our design to support multiple customers.

Defenses: We assume the ISP has a pre-defined *library of defenses* specifying a defense strategy for each attack type. For each attack type A_a, the defense strategy is specified as a directed acyclic graph DAG_a representing a typical multi-stage attack analysis and mitigation procedure. Each node of the graph represents a logical module and the edges are tagged with the result of the previous nodes processing (e.g., "benign" or "attack" or "analyze further"). Each logical node will be realized by one (or more) *virtual appliance(s)* depending on the attack volume. Figure 3 shows an example strategy graph with 4 modules used for defending against a UDP flood attack. Here, the first module tracks the number of UDP packets each source sends and performs a simple threshold-based check to decide whether the source needs to be let through or throttled.

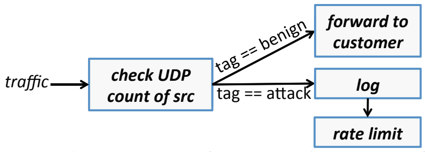


Figure 3: A sample defense against UDP flood.

Our goal here is not to develop new defense algorithms but to develop the system orchestration capabilities to enable flexible and elastic defense. As such, we assume the *DAG*s have been provided by domain experts, DDoS defense vendors, or by consulting best practices.

3.2 Bohatei workflow and challenges

The workflow of Bohatei has four steps (see Figure 2):

1. *Attack detection:* We assume the ISP uses some out-of-band anomaly detection technique to flag whether a customer is under a DDoS attack [27]. The design of this detection algorithm is outside the scope of this paper. The detection algorithm gives a coarse-grained specification of the suspicious traffic, indicating the customer under attack and some coarse identifications of the type and sources of the attack; e.g., "srcprefix=*,dstprefix=cust,type=SYN".
2. *Attack estimation:* Once suspicious traffic is detected, the strategy module estimates the volume of suspicious traffic of each attack type arriving at each ingress.
3. *Resource management:* The resource manager then uses these estimates as well as the library of defenses to determine the type, number, and the location of defense VMs that need to be instantiated. The goal of the resource manager is to efficiently assign available network resources to the defense while minimizing user-perceived latency and network congestion.
4. *Network orchestration:* Finally, the network orchestration module sets up the required network forwarding rules to steer suspicious traffic to the defense VMs as mandated by the resource manager.

Given this workflow, we highlight the three challenges we need to address to realize our vision:

C1. Responsive resource management: We need an efficient way of assigning the ISP's available compute and network resources to DDoS defense. Specifically, we need to decide how many VMs of each type to run on each server of each datacenter location so that attack traffic is handled properly while minimizing the latency experienced by legitimate traffic. Doing so in a *responsive* manner (e.g., within tens of seconds), however, is challenging. Specifically, this entails solving a large NP-hard optimization problem, which can take several hours to solve even with state-of-the-art solvers.

C2. Scalable network orchestration: The canonical view in SDN is to set up switch forwarding rules in a *per-flow* and *reactive* manner [40]. That is, every time a switch receives a flow for which it does not have a forwarding entry, the switch queries the SDN controller to get the forwarding rule. Unfortunately, this per-flow and reactive paradigm is fundamentally unsuitable for DDoS defense. First, an adversary can easily saturate the control plane bandwidth as well as the controller compute resources [54]. Second, installing per-flow rules on the switches will quickly exhaust the limited rule space (≈4K TCAM rules). Note that unlike traffic engineering applications of SDN [34], coarse-grained IP prefix-based forwarding policies would not suffice in the context of DDoS defense, as we cannot predict the IP prefixes of future attack traffic.

C3. Dynamic adversaries: Consider a dynamic adversary who can rapidly change the attack mix (i.e., attack type, volume, and ingress point). This behavior can make the ISP choose between two undesirable choices: (1) wasting compute resources by overprovisioning for attack scenarios that may not ever arrive, (2) not instantiating the required defenses (to save resources), which will let attack traffic reach the customer.

3.3 High-level approach

Next we highlight our key ideas to address C1–C3:

- **Hierarchical optimization decomposition (§4):** To address C1, we use a hierarchical decomposition of the resource optimization problem into two stages. First, the Bohatei global (i.e., ISP-wide) controller uses coarse-grained information (e.g., total spare capacity of each datacenter) to determine how many and what types of VMs to run in each datacenter. Then, each local (i.e., per-datacenter) controller uses more fine-grained information (e.g., location of available servers) to determine the specific server on which each defense VM will run.
- **Proactive tag-based forwarding (§5):** To address C2, we design a scalable orchestration mechanism using two key ideas. First, switch forwarding rules are based on per-VM tags rather than per-flow to dramatically reduce the size of the forwarding tables. Second, we proactively configure the switches to eliminate frequent interactions between the switches and the control plane [40].

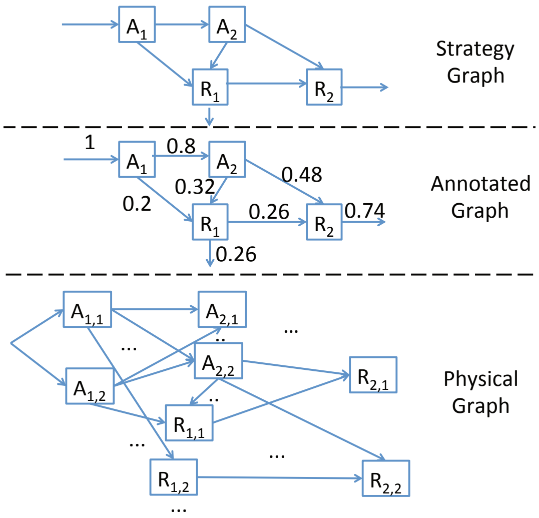


Figure 4: An illustration of strategy vs. annotated vs. physical graphs. Given annotated graphs and suspicious traffic volumes, the resource manager computes physical graphs.

- **Online adaptation (§6):** To handle a dynamic adversary that changes the attack mix (C3), we design a defense strategy adaptation approach inspired by classical online algorithms for regret minimization [36].

4 Resource Manager

The goal of the resource management module is to efficiently determine network and compute resources to analyze and take action on suspicious traffic. The key here is responsiveness—a slow algorithm enables adversaries to nullify the defense by rapidly changing their attack characteristics. In this section, we describe the optimization problem that Bohatei needs to solve and then present a scalable heuristic that achieves near optimal results.

4.1 Problem inputs

Before we describe the resource management problem, we establish the main input parameters: the ISP's compute and network parameters and the defense processing requirements of traffic of different attack types. We consider an ISP composed of a set of edge PoPs[5] $E = \{E_e\}_e$ and a set of datacenters $D = \{D_d\}_d$.

ISP constraints: Each datacenter's traffic processing capacity is determined by a pre-provisioned uplink capacity C_d^{link} and compute capacity $C_d^{compute}$. The compute capacity is specified in terms of the number of VM slots, where each VM slot has a given capacity specification (e.g., instance sizes in EC2 [2]).

Processing requirements: As discussed earlier in §3.1, different attacks require different *strategy graphs*. However, the notion of a strategy graph by itself will not suf-

[5] We use the terms "edge PoP" and "ingress" interchangeably.

fice for resource management, as it is does not specify the traffic volume that at each module should process.

The input to the resource manager is in form of *annotated graphs* as shown in Figure 4. An annotated graph $DAG_a^{annotated}$ is a strategy graph annotated with edge weights, where each weight represents the fraction of the total input traffic to the graph that is expected to traverse the corresponding edge. These weights are precomputed based on prior network monitoring data (e.g., using NetFlow) and from our adaptation module (§6). $T_{e,a}$ denotes the volume of suspicious traffic of type a arriving at edge PoP e. For example, in Figure 4, weight 0.48 from node A_2 to node R_2 means 48% of the total input traffic to the graph (i.e., to A_1) is expected to traverse edge $A_2 \rightarrow R_2$.

Since modules may vary in terms of compute complexity and the traffic rate that can be handled per VM-slot, we need to account for the parameter $P_{a,i}$ that is the traffic processing capacity of a VM (e.g., in terms of compute requirements) for the logical module $v_{a,i}$, where $v_{a,i}$ is node i of graph $DAG_a^{annotated}$.

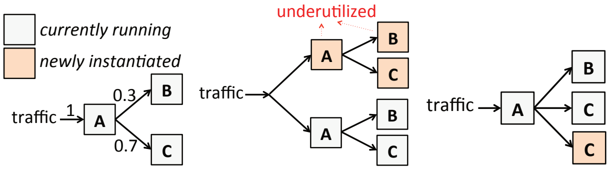
Network footprint: We denote the network-level cost of transferring the unit of traffic from ingress e to datacenter d by $L_{e,d}$; e.g., this can represent the path latency per byte of traffic. Similarly, within a datacenter, the units of intra-rack and inter-rack traffic costs are denoted by *IntraUnitCost* and *InterUnitCost*, respectively (e.g., they may represent latency such that $IntraUnitCost < InterUnitCost$).

4.2 Problem statement

Our resource management problem is to translate the annotated graph into a *physical graph* (see Figure 4); i.e., each node i of the annotated graph $DAG_a^{annotated}$ will be realized by one or more VMs each of which implement the logical module $v_{a,i}$.

Fine-grained scaling: To generate physical graphs given annotated graphs in a resource-efficient manner, we adopt a *fine-grained scaling* approach, where each logical module is scaled independently. We illustrate this idea in Figure 5. Figure 5a shows an annotated graph with three logical modules A, B, and C, receiving different amounts of traffic and consuming different amounts of compute resources. Once implemented as a physical graph, suppose module C becomes the bottleneck due to its processing capacity and input traffic volume. Using a monolithic approach (e.g., running A, B, and C within a single VM), we will need to scale the entire graph as shown in Figure 5b. Instead, we decouple the modules to enable scaling out individual VMs; this yields higher resource efficiency as shown in Figure 5c.

Goals: Our objective here is to (a) instantiate the VMs across the compute servers throughout the ISP, and (b)



(a) Annotated graph. (b) Monolithic. (c) Fine-grained.

Figure 5: An illustration of fine-grained elastic scaling when module C becomes the bottleneck.

distribute the processing load across these servers to minimize the expected latency for legitimate traffic. Further, we want to achieve (a) and (b) while minimizing the footprint of suspicious traffic.[6]

To this end, we need to assign values to two key sets of decision variables: (1) the fraction of traffic $T_{e,a}$ to send to each datacenter D_d (denoted by $f_{e,a,d}$), and (2) the number of VMs of type $v_{a,i}$ to run on server s of datacenter D_d. Naturally, these decisions must respect the datacenters' bandwidth and compute constraints.

Theoretically, we can formulate this resource management problem as a constrained optimization via an Integer Linear Program (ILP). For completeness, we describe the full ILP in Appendix A. Solving the ILP formulation gives an optimal solution to the resource management problem. However, if the ILP-based solution is incorporated into Bohatei, an adversary can easily overwhelm the system. This is because the ILP approach takes several hours (see Table 2). By the time it computes a solution, the adversary may have radically changed the attack mix.

4.3 Hierarchical decomposition

To solve the resource management problem, we decompose the optimization problem into two subproblems: (1) the Bohatei global controller solves a *Datacenter Selection Problem (DSP)* to choose datacenters responsible for processing suspicious traffic, and (2) given the solution to the DSP, each local controller solves a *Server Selection Problem (SSP)* to assign servers inside each selected datacenter to run the required VMs. This decomposition is naturally scalable as the individual SSP problems can be solved independently by datacenter controllers. Next, we describe practical greedy heuristics for the DSP and SSP problems that yield close-to-optimal solutions (see Table 2).

Datacenter selection problem (DSP): We design a greedy algorithm to solve DSP with the goal of reducing ISP-wide suspicious traffic footprint. To this end, the algorithm first sorts suspicious traffic volumes (i.e.,

[6] While it is possible to explicitly minimize network congestion [33], minimizing suspicious traffic footprint naturally helps reduce network congestion as well.

$T_{e,a}$ values) in a decreasing order. Then, for each suspicious traffic volume $T_{e,a}$ from the sorted list, the algorithm tries to assign the traffic volume to the datacenter with the least cost based on $L_{e,d}$ values. The algorithm has two outputs: (1) $f_{e,a,d}$ values denoting what fraction of suspicious traffic from each ingress should be steered to each datacenter (as we will see in §5, these values will be used by network orchestration to steer traffic correspondingly), (2) the physical graph corresponding to attack type a to be deployed by datacenter d. For completeness, we show the pseudocode for the DSP algorithm in Figure 16 in Appendix B.

Server selection problem (SSP): Intuitively, the SSP algorithm attempts to preserve traffic locality by instantiating nodes adjacent in the physical graph as close as possible within the datacenter. Specifically, given the physical graph, the SSP algorithm greedily tries to assign nodes with higher capacities (based on $P_{a,i}$ values) along with its predecessors to the same server, or the same rack. For completeness we show the pseudocode for the SSP algorithm in Figure 17 in Appendix B.

5 Network Orchestration

Given the outputs of the resource manager module (i.e., assignment of datacenters to incoming suspicious traffic and assignment of servers to defense VMs), the role of the network orchestration module is to configure the network to implement these decisions. This includes setting up forwarding rules in the ISP backbone and inside the datacenters. The main requirement is scalability in the presence of attack traffic. In this section, we present our *tag-based* and *proactive* forwarding approach to address the limitations of the per-flow and reactive SDN approach.

5.1 High-level idea

As discussed earlier in §3.2, the canonical SDN view of setting up switch forwarding rules in a per-flow and reactive manner is not suitable in the presence of DDoS attacks. Furthermore, there are practical scalability and deployability concerns with using SDN in ISP backbones [21,29]. There are two main ideas in our approach to address these limitations:

- Following the hierarchical decomposition in resource management, we also decompose the network orchestration problem into two-sub-problems: (1) Wide-area routing to get traffic to datacenters, and (2) Intra-datacenter routing to get traffic to the right VM instances. This decomposition allows us to use different network-layer techniques; e.g., SDN is more suitable inside the datacenter while traditional MPLS-style routing is better suited for wide-area routing.
- Instead of the controller reacting to each flow arrival, we *proactively* install forwarding rules before traffic arrives. Since we do not know the specific IP-level suspicious flows that will arrive in the future, we use logical *tag-based* forwarding rules with per-VM tags instead of per-flow rules.

5.2 Wide-area orchestration

The Bohatei global controller sets up forwarding rules on backbone routers so that traffic detected as suspicious is steered from edge PoPs to datacenters according to the resource management decisions specified by the $f_{e,a,d}$ values (see §4.3).[7]

To avoid a forklift upgrade of the ISP backbone and enable an immediate adoption of Bohatei, we use traditional tunneling mechanisms in the backbone (e.g., MPLS or IP tunneling). We proactively set up static tunnels from each edge PoP to each datacenter. Once the global controller has solved the DSP problem, the controller configures backbone routers to split the traffic according to the $f_{e,a,d}$ values. While our design is not tied to any specific tunneling scheme, the widespread use of MPLS and IP tunneling make them natural candidates [34].

5.3 Intra-datacenter orchestration

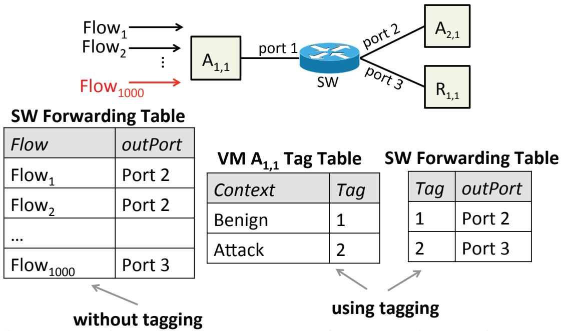
Inside each datacenter, the traffic needs to be steered through the intended sequence of VMs. There are two main considerations here:

1. The next VM a packet needs to be sent to depends on the *context* of the current VM. For example, the node *check UDP count of src* in the graph shown in Figure 3 may send traffic to either *forward to customer* or *log* depending on its analysis outcome.
2. With elastic scaling, we may instantiate several physical VMs for each logical node depending on the demand. Conceptually, we need a “load balancer” at every level of our annotated graph to distribute traffic across different VM instances of a given logical node.

Note that we can trivially address both requirements using a per-flow and reactive solution. Specifically, a local controller can track a packet as it traverses the physical graph, obtain the relevant context information from each VM, and determine the next VM to route the traffic to. However, this approach is clearly not scalable and can introduce avenues for new attacks. The challenge here is to meet these requirements without incurring the overhead of this per-flow and reactive approach.

Encoding processing context: Instead of having the controller track the context, our high-level idea is to en-

[7] We assume the ISP uses legacy mechanisms for forwarding non-attack traffic and traffic to non-Bohatei customers, so these are not the focus of our work.



SW Forwarding Table

Flow	*outPort*
Flow$_1$	Port 2
Flow$_2$	Port 2
…	
Flow$_{1000}$	Port 3

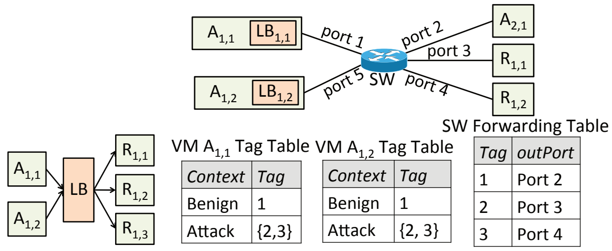
VM A$_{1,1}$ Tag Table

Context	*Tag*
Benign	1
Attack	2

SW Forwarding Table

Tag	*outPort*
1	Port 2
2	Port 3

Figure 6: Context-dependent forwarding using tags.



VM A$_{1,1}$ Tag Table

Context	*Tag*
Benign	1
Attack	{2,3}

VM A$_{1,2}$ Tag Table

Context	*Tag*
Benign	1
Attack	{2, 3}

SW Forwarding Table

Tag	*outPort*
1	Port 2
2	Port 3
3	Port 4

(a) A naive load balancer design.

(b) A distributed load balancer design.

Figure 7: Different load balancer design points.

code the necessary context as *tags* inside packet headers [31]. Consider the example shown in Figure 6 composed of VMs $A_{1,1}$, $A_{2,1}$, and $R_{1,1}$. $A_{1,1}$ encodes the processing context of outgoing traffic as tag values embedded in its outgoing packets (i.e., tag values 1 and 2 denote benign and attack traffic, respectively). The switch then uses this tag value to forward each packet to the correct next VM.

Tag-based forwarding addresses the control channel bottleneck and switch rule explosion. First, the tag generation and tag-based forwarding behavior of each VM and switch is configured proactively once the local controller has solved the SSP. We proactively assign a tag for each VM and populate forwarding rules before flows arrive; e.g., in Figure 6, the tag table of $A_{1,1}$ and the forwarding table of the router have been already populated as shown. Second, this reduces router forwarding rules as illustrated in Figure 6. Without tagging, there will be one rule for each of the 1000 flows. Using tag-based forwarding, we achieve the same forwarding behavior using only two forwarding rules.

Scale-out load balancing: One could interconnect VMs of the same physical graph as shown in Figure 7a using a dedicated load balancer (load balancer). However, such a load balancer may itself become a bottleneck, as it is on the path of every packet from any VM in the set $\{A_{1,1}, A_{1,2}\}$ to any VM in the set $\{R_{1,1}, R_{1,2}, , R_{1,3}\}$. To circumvent this problem, we implement the distribution strategy *inside each VM* so that the load balancer capability scales proportional to the current number of VMs. Consider the example shown in Figure 7b where due to an increase in attack traffic volume we have added one more VM of type A_1 (denoted by $A_{1,2}$) and one more VM of type R_1 (denoted by $R_{1,2}$). To load balance traffic between the two VMs of type R_1, the load balancer of A_1 VMs (shown as $LB_{1,1}$ and $LB_{1,2}$ in the figure) pick a tag value from a *tag pool* (shown by $\{2,3\}$ in the figure) based on the processing context of the outgoing packet and the intended load balancing scheme (e.g., uniformly at random to distribute load equally). Note that this tag pool is pre-populated by the local controller (given the defense library and the output of the resource manager module). This scheme, thus, satisfies the load balancing requirement in a scalable manner.

Other issues: There are two remaining practical issues:

- *Number of tag bits:* We give a simple upper bound on the required number of bits to encode tags. First, to support context-dependent forwarding out of a VM with k relevant contexts, we need k distinct tag values. Second. to support load balancing among l VMs of the same logical type, each VM needs to be populated with a tag pool including l tags. Thus, at each VM we need at most $k \times l$ distinct tag values. Therefore, an upper bound on the total number of unique tag values is $k_{max} \times l_{max} \times \sum_a | V_a^{annotated} |$, where k_{max} and l_{max} are the maximum number of contexts and VMs of the same type in a graph, and $V_a^{annotated}$ is the set of vertices of annotated graph for attack type a. To make this concrete, across the evaluation experiments §8, the maximum value required tags was 800, that can be encoded in $\lceil log_2(800) \rceil = 10$ bits. In practice, this tag space requirement of Bohatei can be easily satisfied given that datacenter grade networking platforms already have extensible header fields [56].
- *Bidirectional processing:* Some logical modules may have bidirectional semantics. For example, in case of a DNS amplification attack, request and response traffic must be processed by the same VM. (In other cases, such as the UDP flood attack, bidirectionality is not required.). To enforce bidirectionality, ISP edge switches use tag values of outgoing traffic so that when the corresponding incoming traffic comes back, edge switches sends it to the datacenter within which the VM that processed the outgoing traffic is located. Within the datacenter, using this tag value, the traffic is steered to the VM.

6 Strategy Layer

As we saw in §4, a key input to the resource manager module is the set of $T_{e,a}$ values, which represents the volume of suspicious traffic of each attack type a arriving at each edge PoP e. This means we need to estimate the future attack mix based on observed measurements of the network and then instantiate the required defenses. We begin by describing an adversary that intends to thwart a Bohatei-like system. Then, we discuss limitations of strawman solutions before describing our *online adaptation* mechanism.

Interaction model: We model the interaction between the ISP running Bohatei and the adversary as a repeated interaction over several *epochs*. The ISP's "move" is one epoch behind the adversary; i.e., it takes Bohatei an epoch to react to a new attack scenario due to implementation delays in Bohatei operations. The epoch duration is simply the sum of the time to detect the attack, run the resource manager, and execute the network orchestration logic. While we can engineer the system to minimize this lag, there will still be non-zero delays in practice and thus we need an adaptation strategy.

Objectives: Given this interaction model, the ISP has to pre-allocate VMs and hardware resources for a specific attack mix. An intelligent and dynamic adversary can change its attack mix to meet two goals:

G1 *Increase hardware resource consumption:* The adversary can cause ISP to overprovision defense VMs. This may impact the ISP's ability to accommodate other attack types or reduce profits from other services that could have used the infrastructure.

G2 *Succeed in delivering attack traffic:* If the ISP's detection and estimation logic is sub-optimal and does not have the required defenses installed, then the adversary can maximize the volume of attack traffic delivered to the target.

The adversary's goal is to maximize these objectives, while the ISPs goal is to minimize these to the extent possible. One could also consider a third objective of collateral damage on legitimate traffic; e.g., introduce needless delays. We do not discuss this dimension because our optimization algorithm from §4 will naturally push the defense as close to the ISP edge (i.e., traffic ingress points) as possible to minimize the impact on legitimate traffic.

Threat model: We consider an adversary with a fixed budget in terms of the total volume of attack traffic it can launch at any given time. Note that the adversary can apportion this budget across the *types* of attacks and the *ingress* locations from which the attacks are launched. Formally, we have $\sum_{e}\sum_{a} T_{e,a} \leq B$, but there are no constraints on the specific $T_{e,a}$ values.

Limitations of strawman solutions: For simplicity, let us consider a single ingress point. Let us consider a strawman solution called *PrevEpoch* where we measure the attack observed in the previous epoch and use it as the estimate for the next epoch. Unfortunately, this can have serious issues w.r.t. goals **G1** and **G2**. To see why, consider a simple scenario where we have two attack types with a budget of 30 units and three epochs with the attack volumes as follows: T1: A1= 10, A2=0; T2: A1=20, A2=0; T3: A1=0; A2=30. Now consider the *PrevEpoch* strategy starting at the 0,0 configuration. It has a total wastage of 0,0,20 units and a total evasion of 10,10,30 units because it has overfit to the previous measurement. We can also consider other strategies; e.g., a *Uniform* strategy that provisions 15 units each for A1 and A2 or extensions of these to *overprovision* where we multiply the number of VMs given by the resource manager in the last epoch by a fixed value $\gamma > 1$. However, these suffer from the same problems and are not competitive.

Online adaptation: Our metric of success here is to have *low regret* measured with respect to the best static solution computed in hindsight [36]. Note that in general, it is not possible to be competitive w.r.t. the best dynamic solution since that presumes oracle knowledge of the adversary, which is not practical.

Intuitively, if we have a non-adaptive adversary, using the observed empirical average is the best possible static hindsight estimation strategy; i.e., $T^{*}_{e,a} = \frac{\sum_t T_{e,a,t}}{|t|}$ would be the optimal solution ($|t|$ denotes the total number of epochs). However, an attacker who knows that we are using this strategy can game the system by changing the attack mix. To address this, we use a follow the perturbed leader (FPL) strategy [36] where our estimation uses a combination of the past observed behavior of the adversary and a randomized component. Intuitively, the random component makes it impossible for the attacker to predict the ISP's estimates. This is a well-known approach in online algorithms to minimize the regret [36]. Specifically, the traffic estimates for the *next* epoch $t+1$, denoted by $\widehat{T_{e,a,t+1}}$ values, are calculated based on the average of the past values plus a random component: $\widehat{T_{e,a,t+1}} = \frac{\sum_{t'=1}^{t} T_{e,a,t'}}{|t|} + randperturb$.

Here, $T_{e,a,t'}$ is the empirically observed value of the attack traffic and *randperturb* is a random value drawn uniformly from $[0, \frac{2 \times B}{nextEpoch \times |E| \times |A|}]$. (This is assuming a total defense of budget of $2 \times B$.) It can be shown that this is indeed a provably good regret minimization strategy [36]; we do not show the proof for brevity.

7 Implementation

In this section, we briefly describe how we implemented the key functions described in the previous sections. We have made the source code available [1].

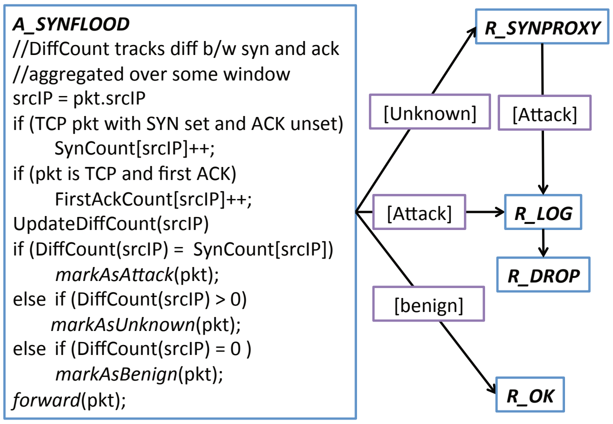
7.1 DDoS defense modules

The design of the Bohatei strategy layer is inspired by the prior modular efforts in Click [7] and Bro [46]. This modularity has two advantages. First, it allows us to adopt best of breed solutions and compose them for different attacks. Second, it enables more fine-grained scaling. At a high level, there are two types of logical building blocks in our defense library:

1. *Analysis (A)*: Each analysis module processes a suspicious flow and determines appropriate action (e.g., more analysis or specific response). It receives a packet and outputs a tagged packet, and the tags are used to steer traffic to subsequent analysis and response module instances as discussed earlier.
2. *Response (R)*: The input to an R module is a tagged packet from some A module. Typical responses include forward to customer (for benign traffic), log, drop, and rate limit. Response functions will depend on the type of attack; e.g., sending RST packets in case of a TCP SYN attack.

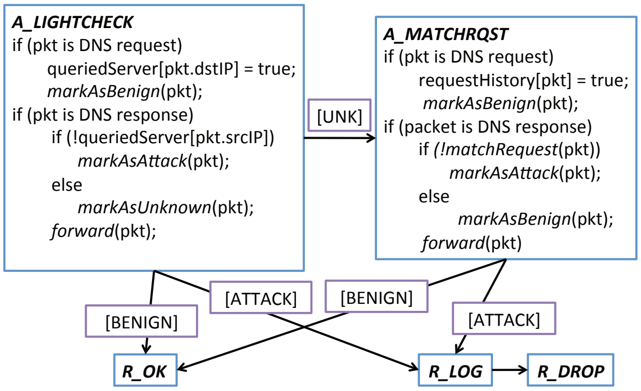
Next, we describe defenses we have implemented for different DDoS attacks. Our goal here is to illustrate the flexibility Bohatei provides in dealing with a diverse set of known attacks rather than develop new defenses.

1. **SYN flood** (Figure 8): We track the number of open TCP sessions for each source IP; if a source IP has no asymmetry between SYNs and ACKs, then mark its packets as benign. If a source IP never completes a connection, then we can mark its future packets as known attack packets. If we see a gray area where the source IP has completed some connections but not others, in which case we use a SYN-Proxy defense (e.g., [9, 28]).
2. **DNS amplification** (Figure 9): We check if the DNS server has been queried by some customer IP. This example highlights another advantage—we can decouple fast (e.g., the header-based *A_LIGHTCHECK* module) and slow path analyses (e.g., the second A module needs to look into payloads). The responses are quite simple and implement logging, dropping, or basic forwarding to the destination. We do not show the code for brevity.
3. **UDP flood:** The analysis node *A_UDP* identifies source IPs that send an anomalously higher number of UDP packets and uses this to categorize each packet as either attack or benign. The function *forward* will direct the packet to the next node in the defense strategy; i.e., *R_OK* if benign, or *R_LOG* if attack.
4. **Elephant flow:** Here, the attacker launches legitimate but very large flows. The A module detects abnormally large flows and flags them as attack flows. The response is to randomly drop packets from these large flows (not shown).



```
A_SYNFLOOD
//DiffCount tracks diff b/w syn and ack
//aggregated over some window
srcIP = pkt.srcIP
if (TCP pkt with SYN set and ACK unset)
      SynCount[srcIP]++;
if (pkt is TCP and first ACK)
      FirstAckCount[srcIP]++;
UpdateDiffCount(srcIP)
if (DiffCount(srcIP) =  SynCount[srcIP])
      markAsAttack(pkt);
else  if (DiffCount(srcIP) > 0)
     markAsUnknown(pkt);
else  if (DiffCount(srcIP) = 0 )
      markAsBenign(pkt);
forward(pkt);
```

Figure 8: SYN Flood defense strategy graph.



```
A_LIGHTCHECK
if (pkt is DNS request)
      queriedServer[pkt.dstIP] = true;
      markAsBenign(pkt);
if (pkt is DNS response)
      if (!queriedServer[pkt.srcIP])
          markAsAttack(pkt);
      else
          markAsUnknown(pkt);
      forward(pkt);
```

```
A_MATCHRQST
if (pkt is DNS request)
      requestHistory[pkt] = true;
      markAsBenign(pkt);
if (packet is DNS response)
      if (!matchRequest(pkt))
          markAsAttack(pkt);
      else
          markAsBenign(pkt);
      forward(pkt)
```

Figure 9: DNS amplification defense strategy graph.

Attack detection: We use simple time series anomaly detection using `nfdump`, a tool that provides NetFlow-like capabilities, and custom code [27]. The output of the detection module is sent to the Bohatei global controller as a 3-tuple $\langle Type, FlowSpec, Volume \rangle$, where *Type* indicates the type of DDoS attack (e.g., SYN flood, DNS amplification), *FlowSpec* provides a generic description of the *flow space* of suspicious traffic (involving wildcards), and *Volume* indicates the suspicious traffic volume based on the flow records. Note that this *FlowSpec* does not pinpoint specific attack flows; rather, it is a coarse-grained hint on characteristics of suspicious traffic that need further processing through the defense graphs.

7.2 SDN/NFV platform

Control plane: We use the `OpenDayLight` network control platform, as it has gained significant traction from key industry players [17]. We implemented the Bohatei global and local control plane modules (i.e., strategy, resource management, and network orchestration) as separate `OpenDayLight` plugins. Bohatei uses OpenFlow [40] for configuring switches; this is purely for ease of prototyping, and it is easy to integrate other network control APIs (e.g., YANG/NetCONF).

Data plane: Each physical node is realized using a VM running on KVM. We use open source tools (e.g., Snort, Bro) to implement the different Analysis (A) and

Attack type	Analysis	Response
UDP flood	A_UDP using Snort (inline mode)	R_LOG using iptables and R_RATELIMIT using tc library
DNS amp.	both LIGHTCHECK and MATCHRQST using netfilter library, iptables, custom code	R_LOG and R_DROP using iptables
SYN flood	A_SYNFLOOD using Bro	R_SYNPROXY using PF firewall, R_LOG and R_DROP using iptables
Elephant flow	A_ELEPHANT using netfilter library, iptables, custom code	R_DROP using iptables

Table 1: Implementation of Bohatei modules.

Response (R) modules. Table 1 summarizes the specific platforms we have used. These tools are instrumented using FlowTags [31] to add tags to outgoing packets to provide contextual information. We used OpenvSwitch [16] to emulate switches in both datacenters and ISP backbone. The choice of OpenvSwitch is for ease of prototyping on our testbed.

Resource management algorithms: We implement the DSP and SSP algorithms using custom `Go` code.

8 Evaluation

In this section, we show that:

1. Bohatei is scalable and handles attacks of hundreds of Gbps in large ISPs and that our design decisions are crucial for its scale and responsiveness (§8.1)
2. Bohatei enables a rapid (≤ 1 minute) response for several canonical DDoS attack scenarios (§8.2)
3. Bohatei can successfully cope with several dynamic attack strategies (§8.3)

Setup and methodology: We use a combination of real testbed and trace-driven evaluations to demonstrate the above benefits. Here we briefly describe our testbed, topologies, and attack configurations:

- *SDN Testbed:* Our testbed has 13 Dell R720 machines (20-core 2.8 GHz Xeon CPUs, 128GB RAM). Each machine runs KVM on CentOS 6.5 (Linux kernel v2.6.32). On each machine, we assigned equal amount of resources to each VM: 1 vCPU (virtual CPU) and 512MB of memory.
- *Network topologies:* We emulate several router-level ISP topologies (6–196 nodes) from the Internet Topology Zoo [22]. We set the bandwidth of each core link to be 100Gbps and link latency to be 10ms. The number of datacenters, which are located randomly, is 5% of the number of backbone switches with a capacity of 4,000 VMs per datacenter.
- *Benign traffic demands:* We assume a gravity model of traffic demands between ingress-egress switch pairs [50]. The total volume is scaled linearly with the size of the network such that the average link load on the topology backbone is 24Gbps with a maximum bottleneck link load of 55Gbps. We use `iperf` and custom code to generate benign traffic.
- *Attack traffic:* We implemented custom modules to generate attack traffic: (1) **SYN flood** attack by sending only SYN packets with spoofed IP addresses at a high rate; (2) **DNS amplification** using `OpenDNS` server with `BIND` (version 9.8) and emulating an attacker sending DNS requests with spoofed source IPs; (3) We use `iperf` to create some fixed bandwidth traffic to generate **elephant flows**, and (4) **UDP flood** attacks. We randomly pick one edge PoP as the target and vary the target across runs. We ramp up the attack volume until it induces maximum reduction in throughput of benign flows to the target. On our testbed, we can ramp up the volume up to 10 Gbps. For larger attacks, we use simulations.

Topology	#Nodes	Run time (secs)		Optimality
		Baseline	Bohatei	Gap
Heanet	6	205	0.002	0.0003
OTEGlobe	92	2234	0.007	0.0004
Cogent	196	> 1 hr	0.01	0.0005

Table 2: Run time and optimality gap of Bohatei vs. ILP formulation across different topologies.

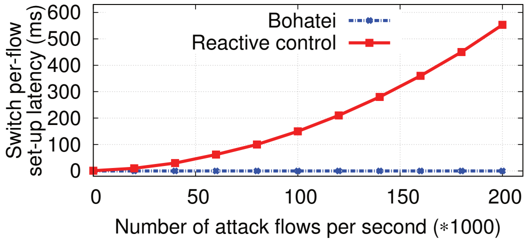


Figure 10: Bohatei control plane scalability.

8.1 Bohatei scalability

Resource management: Table 2 compares the run time and optimality of the ILP-based algorithm and Bohatei (i.e., DSP and SSP) for 3 ISP topologies of various sizes. (We have results for several other topologies but do not show it for brevity.) The ILP approach takes from several tens of minutes to hours, whereas Bohatei takes only a few milliseconds enabling rapid response to changing traffic patterns. The optimality gap is $\leq 0.04\%$.

Control plane responsiveness: Figure 10 shows the per-flow setup latency comparing Bohatei to the SDN per-flow and reactive paradigm as the number of attack flows in a DNS amplification attack increases. (The results are consistent for other types of attacks and are not shown for brevity.) In both cases, we have a dedicated machine for the controller with 8 2.8GHz cores and 64

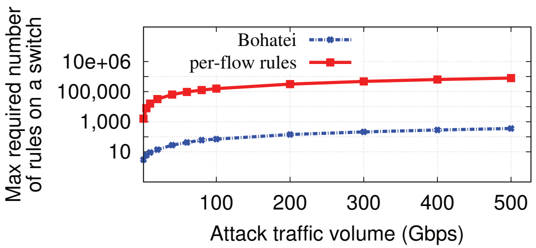


Figure 11: Number of switch forwarding rules in Bohatei vs. today's flow-based forwarding.

GB RAM. To put the number of flows in context, 200K flows roughly corresponds to a 1 Gbps attack. Note that a typical upper bound for switch flow set-up time is on the order of a few milliseconds [59]. We see that Bohatei incurs zero rule setup latency, while the reactive approach deteriorates rapidly as the attack volume increases.

Number of forwarding rules: Figure 11 shows the maximum number of rules required on a switch across different topologies for the SYN flood attack. Using today's flow-based forwarding, each new flow will require a rule. Using tag-based forwarding, the number of rules depends on the number of VM instances, which reduces the switch rule space by four orders of magnitude. For other attack types, we observed consistent results (not shown). To put this in context, the typical capacity of an SDN switch is 3K-4K rules (shared across various network management tasks). This means that per-flow rules will not suffice for attacks beyond 10Gbps. In contrast, Bohatei can handle hundreds of Gbps of attack traffic; e.g., a 1 Tbps attack will require $<$ 1K rules on a switch.

Benefit of scale-out load balancing: We measured the resources that would be consumed by a dedicated load balancing solution. Across different types of attacks with a fixed rate of 10Gbps, we observed that a dedicated load balancer design requires between 220–300 VMs for load balancing alone. By delegating the load balancing task to the VMs, our design obviates the need for these extra load balancers (not shown).

8.2 Bohatei end-to-end effectiveness

We evaluated the effectiveness of Bohatei under four different types of DDoS attacks. We launch the attack traffic of the corresponding type at 10th second; the attack is sustained for the duration of the experiment. In each scenario, we choose the attack volume such that it is capable of bringing the throughput of the benign traffic to zero. Figure 12 shows the impact of attack traffic on the throughput of benign traffic. The Y axis for each scenario shows the network-wide throughput for TCP traffic (a total of 10Gbps if there is no attack). The results shown in this figure are based on Cogent, the largest topology with 196 switches; the results for other topologies were consistent and are not shown. While we do see some small differences across attacks, the overall reaction time is short.

Attack type	# VMs needed	
	Monolithic	Fine-grained scaling
DNS Amplification	5,422	1,005
SYN Flood	3,167	856
Elephant flows	1,948	910
UDP flood	3,642	1,253

Table 3: Total hardware provisioning cost needed to handle a 100 Gbps attack for different attacks.

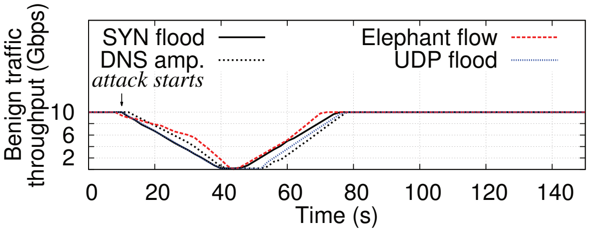


Figure 12: Bohatei enables rapid response and restores throughput of legitimate traffic.

The key takeaway is that Bohatei can help networks respond rapidly (within one minute) to diverse attacks and restore the performance of legitimate flows. We repeated the experiments with UDP as the benign traffic. In this case, the recovery time was even shorter, as the throughput does not suffer from the congestion control mechanism of TCP.

Hardware cost: We measure the total number of VMs needed to handle a given attack volume and compare two cases: (1) monolithic VMs embedding the entire defense logic for an attack, and (2) using Bohatei's fine-grained modular scaling. Table 3 shows the number of VMs required to handle different types of 100 Gbps attacks. Fine-grained scaling gives a 2.1–5.4× reduction in hardware cost vs. monolithic VMs. Assuming a commodity server costs $3,000 and can run 40VMs in Bohatei (as we did), we see that it takes a total hardware cost of less than about $32,000 to handle a 100 Gbps attack across Table 3. This is in contrast to the total server cost of about $160,000 for the same scenario if we use monolithic VMs. Moreover, since Bohatei is horizontally scalable by construction, dealing with larger attacks simply entails a linearly scale up of the number of VMs.

Routing efficiency: To quantify how Bohatei addresses the routing inefficiency of existing solutions (§2.2), we ran the following experiment. For each topology, we measured the end-to-end latency in two equivalently provisioned scenarios: (1) the location of the DDoS defense appliance is the node with the highest betweenness value[8], and (2) Bohatei. As a baseline, we consider

[8] Betweenness is a measure of a node's centrality, which is the fraction of the network's all-pairs shortest paths that pass through that node.

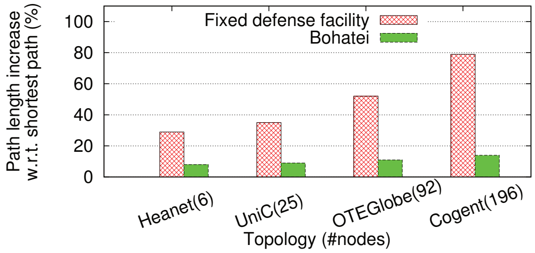
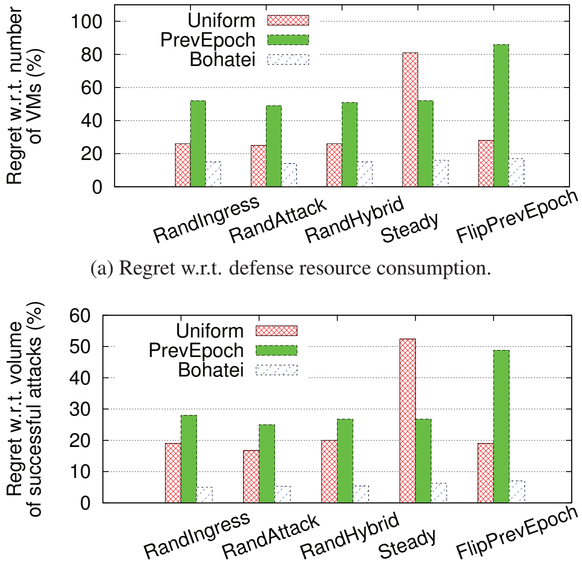


Figure 13: Routing efficiency in Bohatei.



(a) Regret w.r.t. defense resource consumption.

(b) Regret w.r.t. successful attacks.

Figure 14: Effect of different adaptation strategies (bars) vs. different attacker strategies (X axis).

shortest path routing without attacks. The main conclusion in Figure 13 is that Bohatei reduces traffic latency by 20% to 65% across different scenarios.

8.3 Dynamic DDoS attacks

We consider the following dynamic DDoS attack strategies: (1) *RandIngress*: In each epoch, pick a random subset of attack ingresses and distribute the attack budget evenly across attack types; (2) *RandAttack*: In each epoch, pick a random subset of attack types and distribute the budget evenly across all ingresses; (3) *RandHybrid*: In each epoch, pick a random subset of ingresses and attack types independently and distribute the attack budget evenly across selected pairs; (4) *Steady*: The adversary picks a random attack type and a subset of ingresses and sustains it during all epochs; and (5) *FlipPrevEpoch*: This is conceptually equivalent to conducting two *Steady* attacks $A1$ and $A2$ with each being active during odd and even epochs, respectively.

Given the typical DDoS attack duration ($\approx$ 6 hours [12]), we consider an attack lasting for 5000 5-second epochs (i.e., $\approx$7 hours). Bohatei is initialized with a zero starting point of attack estimates. The metric of interest we report is the *normalized regret* with respect to the best static decision in hindsight; i.e., if we had to pick a single static strategy for the entire duration. Figure 14a and Figure 14b show the regret w.r.t. the two goals G1 (the number of VMs) and G2 (volume of successful attack) for a 24-node topology. The results are similar using other topologies and are not shown here. Overall, Bohatei's online adaptation achieves low regret across the adversarial strategies compared to two strawman solutions: (1) uniform estimates, and (2) estimates given the previous measurements.

9 Related Work

DDoS has a long history; we refer readers to surveys for a taxonomy of DDoS attacks and defenses (e.g., [41]). We have already discussed relevant SDN/NFV work in the previous sections. Here, we briefly review other related topics.

Attack detection: There are several algorithms for detecting and filtering DDoS attacks. These include time series detection techniques (e.g., [27]), use of backscatter analysis (e.g., [42]), exploiting attack-specific features (e.g., [35]), and network-wide analysis (e.g., [38]). These are orthogonal to the focus of this paper.

DDoS-resilient Internet architectures: These include the use of capabilities [58], better inter-domain routing (e.g., [60]), inter-AS collaboration (e.g., [39]), packet marking and unforgeable identifiers (e.g., [26]), and traceback (e.g., [51]). However, they do not provide an immediate deployment path or resolution for current networks. In contrast, Bohatei focuses on a more practical, single-ISP context, and is aligned with economic incentives for ISPs and their customers.

Overlay-based solutions: There are overlay-based solutions (e.g., [25,52]) that act as a "buffer zone" between attack sources and targets. The design contributions in Bohatei can be applied to these as well.

SDN/NFV-based security: There are few efforts in this space such as FRESCO [53] and AvantGuard [54]. As we saw earlier, these SDN solutions will introduce new DDoS avenues because of the per-flow and reactive model [54]. Solving this control bottleneck requires hardware modifications to SDN switches to add "stateful" components, which is unlikely to be supported by switch vendors soon [54]. In contrast, Bohatei chooses a proactive approach of setting up tag-based forwarding rules that is immune to these pitfalls.

10 Conclusions

Bohatei brings the flexibility and elasticity benefits of recent networking trends, such as SDN and NFV, to DDoS defense. We addressed practical challenges in the design of Bohatei's resource management algorithms and control/data plane mechanisms to ensure that these do not become bottlenecks for DDoS defense. We implemented a full-featured Bohatei prototype built on industry-standard SDN control platforms and commodity network appliances. Our evaluations on a real testbed show that Bohatei (1) is scalable and responds rapidly to attacks, (2) outperforms naive SDN implementations that do not address the control/data plane bottlenecks, and (3) enables resilient defenses against dynamic adversaries. Looking forward, we believe that these design principles can also be applied to other aspects of network security.

Acknowledgments

This work was supported in part by grant number N00014-13-1-0048 from the Office of Naval Research, and NSF awards 1409758, 1111699, 1440056, and 1440065. Seyed K. Fayaz was supported in part by the CMU Bertucci Fellowship. We thank Limin Jia, Min Suk Kang, the anonymous reviewers, and our shepherd Patrick Traynor for their helpful suggestions.

References

[1] Bohatei. https://github.com/ddos-defense/bohatei.

[2] Amazon EC2. http://aws.amazon.com/ec2/.

[3] Arbor Networks, worldwide infrastructure security report, volume IX, 2014. http://bit.ly/1R0NDRi.

[4] AT&T and Intel: Transforming the Network with NFV and SDN. https://www.youtube.com/watch?v=F55pHxTeJLc#t=76.

[5] AT&T Denial of Service Protection. http://soc.att.com/1IIlUec.

[6] AT&T Domain 2.0 Vision White Paper. http://soc.att.com/1kAw1Kp.

[7] Click Modular Router. http://www.read.cs.ucla.edu/click/click.

[8] CloudFlare. https://www.cloudflare.com/ddos.

[9] DDoS protection using Netfilter/iptables. http://bit.ly/1IImM2F.

[10] Dell PowerEdge Rack Servers. http://dell.to/1dtP5Jk.

[11] GSA Advantage. http://1.usa.gov/1ggEgFN.

[12] Incapsula Survey : What DDoS Attacks Really Cost Businesses, 2014. http://bit.ly/1CFZyIr.

[13] iptables. http://www.netfilter.org/projects/iptables/.

[14] NTP attacks: Welcome to the hockey stick era. http://bit.ly/1ROlwQe.

[15] ONS 2014 Keynote: John Donovan, Senior EVP, AT&T Technology & Network Operations. http://bit.ly/1RQFMko.

[16] Open vSwitch. http://openvswitch.org/.

[17] OpenDaylight project. http://www.opendaylight.org/.

[18] Packet processing on Intel architecture. http://intel.ly/1efIEu6.

[19] Prolexic. http://www.prolexic.com/.

[20] Radware. http://www.radware.com/Solutions/Security/.

[21] Time for an SDN Sequel? http://bit.ly/1BSpdma.

[22] Topology Zoo. www.topology-zoo.org.

[23] Verizon-Carrier Adoption of Software-defined Networking. https://www.youtube.com/watch?v=WVczl03edi4.

[24] ZScaler Cloud Security. http://www.zscaler.com.

[25] D. G. Andersen. Mayday: Distributed filtering for internet services. In *Proc. USITS*, 2003.

[26] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (AIP). In *Proc. SIGCOMM*, 2008.

[27] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Proc. ACM SIGCOMM Workshop on Internet Measurement*, 2002.

[28] R. Cáceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: The devil is in the details. *SIGMETRICS Perform. Eval. Rev.*, 26(3):11–15, Dec. 1998.

[29] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: A retrospective on evolving sdn. In *Proc. HotSDN*, 2012.

[30] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks. In *Proc. IMC*, 2014.

[31] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proc. NSDI*, 2014.

[32] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4D approach to network control and management. *ACM CCR*, 2005.

[33] V. Heorhiadi, S. K. Fayaz, M. Reiter, and V. Sekar. Frenetic: A network programming language. *Information Systems Security*, 2014.

[34] Jain et al. B4: Experience with a globally-deployed software defined wan. In *Proc. SIGCOMM*, 2013.

[35] C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: An effective defense against spoofed ddos traffic. In *Proc. CCS*, 2003.

[36] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 2005.

[37] M. S. Kang, S. B. Lee, and V. Gligor. The crossfire attack. In *Proc. IEEE Security and Privacy*, 2013.

[38] A. Lakhina, M. Crovella, and C. Diot. Mining Anomalies Using Traffic Feature Distributions. In *Proc. SIGCOMM*, 2005.

[39] R. Mahajan et al. Controlling high bandwidth aggregates in the network. *CCR*, 2001.

[40] N. McKeown et al. OpenFlow: enabling innovation in campus networks. *CCR*, March 2008.

[41] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. In *CCR*, 2004.

[42] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 2006.

[43] Network functions virtualisation – introductory white paper. http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[44] A. Networks. ATLAS Summary Report: Global Denial of Service. http://atlas.arbor.net/summary/dos.

[45] P. Patel et al. Ananta: cloud scale load balancing. In *Proc. ACM SIGCOMM*, 2013.

[46] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, 1999.

[47] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe. Arrakis: The operating system is the control plane. In *Proc. OSDI*, 2014.

[48] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *LISA*, 1999.

[49] C. Rossow. Amplification hell: Revisiting network protocols for ddos abuse. In *Proc. USENIX Security*, 2014.

[50] M. Roughan. Simplifying the Synthesis of Internet Traffic Matrices. *ACM SIGCOMM CCR*, 2005.

[51] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *Proc. SIGCOMM*, 2000.

[52] E. Shi, I. Stoica, D. Andersen, and A. Perrig. OverDoSe: A generic DDoS protection service using an overlay network. Technical Report CMU-CS-06-114, School of Computer Science, Carnegie Mellon University, 2006.

[53] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. FRESCO: Modular composable security services for software-defined networks. In *Proc. NDSS*, 2013.

[54] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks. In *Proc. CCS*, 2013.

[55] A. Studer and A. Perrig. The coremelt attack. In *Proc. ESORICS*, 2009.

[56] T. Koponen et al. Network virtualization in multi-tenant datacenters. In *Proc. NSDI*, 2014.

[57] P. Verkaik, D. Pei, T. Schollf, A. Shaikh, A. C. Snoeren, and J. E. van der Merwe. Wresting Control from BGP: Scalable Fine-grained Route Control. In *Proc. USENIX ATC*, 2007.

[58] X. Yang, D. Wetherall, and T. Anderson. A dos-limiting network architecture. In *Proc. SIGCOMM*, 2005.

[59] S. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *Communications Magazine, IEEE*, 2013.

[60] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. In *Proc. IEEE Security and Privacy*, 2011.

A ILP Formulation

The ILP formulation for an optimal resource management (mentioned in §4.2) is shown in Figure 15.

Vairables: In addition to the parameters and variables that we have defined earlier in §4, we define the binary variable $q_{d,a,i,vm,s,i',vm',s',l}$ as follows: if it is 1, VM *vm* of

1 Minimize $\alpha \times \sum_e \sum_a \sum_d f_{e,a,d} \times T_{e,a} \times L_{e,d} + \sum_d dsc_d$

s.t.

2 $\forall e, a : \sum_d f_{e,a,d} = 1 \triangleright$ all suspicious traffic should be served

3 $\forall a, d : t_{a,d} = \sum_e f_{e,a,d} \times T_{e,a} \triangleright$ traffic of each type to each datacenter

4 $\forall d : \sum_a t_{a,d} \leq C_d^{link} \triangleright$ datacenter link capacity

5 $\forall d, a, i : \sum_{s \in S_d} n_{a,i}^{d,s} \geq t_{a,d} \times \frac{\sum_{i' : (i',i) = e^{annotated}_{a,i' \to i}} W_{a,i' \to i}}{P_{a,i}} \triangleright$ provisioning sufficient VMs (S_d is the set of d's servers.)

6 $\forall d, s \in S_d : \sum_a \sum_i n_{a,i}^{d,s} \leq C_{d,s}^{compute} \triangleright$ server compute capacity

7 $\forall d : dsc_d = intraR_d \times IntraUnitCost + interR_d \times InterUnitCost \triangleright$ total cost within each datacenter

8 $\forall d : intraR_d = \sum_a \sum_{(i,i') = e^{annotated}_{a,i \to i'}} \sum_{(s,s') \in sameRack} \sum_{vm=1}^{MaxVM} \sum_{vm'=1}^{MaxVM} \sum_{l=1}^{MaxVol} q_{d,a,i,vm,s,i',vm',s',l} \triangleright$ intra-rack cost

9 $\forall d : interR_d = \sum_a \sum_{(i,i') = e^{annotated}_{a,i \to i'}} \sum_{(s,s') \notin sameRack} \sum_{vm=1}^{MaxVM} \sum_{vm'=1}^{MaxVM} \sum_{l=1}^{MaxVol} q_{d,a,i,vm,s,i',vm',s',l} \triangleright$ inter-rack cost

10 $\forall d, a, i', vm' : \sum_s \sum_{s'} \sum_{i : (i,i') = e^{annotated}_{a,i \to i'}} \sum_{vm=1}^{MaxVM} \sum_{l=1}^{MaxVol} q_{d,a,i,vm,s,i',vm',s',l} \leq P_{a,i'} \triangleright$ enforcing VMs capacities

11 $\forall d, s \in S_d, a, i' : n_{a,i'}^{d,s} \times P_{a,i'} \geq \sum_{vm=1}^{MaxVM} \sum_{vm'=1}^{MaxVM} \sum_{i : (i,i') = e^{annotated}_{a,i \to i'}} \sum_{s'} \sum_{l=1}^{MaxVol} q_{d,a,i,vm,s,i',vm',s',l} \triangleright$ bound traffic volumes

12 $\forall d, s \in S_d, a, i' : n_{a,i'}^{d,s} \times P_{a,i'} \leq \sum_{vm=1}^{MaxVM} \sum_{vm'=1}^{MaxVM} \sum_{i : (i,i') = e^{annotated}_{a,i \to i'}} \sum_{s'} \sum_{l=1}^{MaxVol} q_{d,a,i,vm,s,i',vm',s',l} + 1 \triangleright$ bound traffic volumes

13 $\triangleright$ flow conservation for VM *vm* of type logical node *k* that has both predecessor(s) and successor(s)

$\forall d, a, k, vm : \sum_{vm'=1}^{MaxVM} \sum_{g : (g,k) = e^{annotated}_{a,g \to k}} \sum_s \sum_{s'} \sum_{l=1}^{MaxVol} q_{d,a,g,vm',s',k,vm,s,l} = \sum_{vm'=1}^{MaxVM} \sum_{h : (k,h) = e^{annotated}_{a,k \to h}} \sum_s \sum_{s'} \sum_{l=1}^{MaxVol} q_{d,a,k,vm,s,h,vm',s',l}$

14 $\forall link \in ISP\ backbone : \sum_{link \in Path_{e \to d}} \sum_a f_{e,a,d} \times T_{e,a} \leq \beta \times MaxLinkCapacity \triangleright$ per-link traffic load control

15 $f_{e,a,d} \in [0,1], q_{d,a,i,vm,s,i',vm',s',l} \in \{0,1\}, n_{a,i}^d, n_{a,i}^{d,s} \in \{0,1,\dots\}, t_{a,d}, interR_d, intraR_d, dsc_d \in \mathbb{R} \triangleright$ variables

Figure 15: ILP formulation for an optimal resource management.

type $v_{a,i}$ runs on server s and sends 1 unit of traffic (e.g., 1 Gbps) to VM vm' of type $v_{a,i'}$ that runs on server s', where $e^{annotated}_{a,i \to i'} \in E^{annotated}_a$, and servers s and s' are located in datacenter d; otherwise, $q_{d,a,i,vm,s,i',vm',s',l} = 0$. Here l is an auxiliary subscript indicating that the one unit of traffic associated with q is the lth one out of $MaxVol$ possible units of traffic. The maximum required number of VMs of any type is denoted by $MaxVM$.

The ILP involves two key decision variables: *(1)* $f_{e,a,d}$ is the fraction of traffic $T_{e,a}$ to send to datacenter D_d, and *(2)* $n^{d,s}_{a,i}$ is the number of VMs of type $v_{a,i}$ on server s of datacenter d, hence physical graphs $DAG^{physical}_a$.

Objective function: The objective function (1) is composed of inter-datacenter and intra-datacenter costs, where constant $\alpha > 0$ reflects the relative importance of inter-datacenter cost to intra datacenter cost.

Constraints: Equation (2) ensures all suspicious traffic will be sent to data centers for processing. Equation (3) computes the amount of traffic of each attack type going to each datacenter, which is ensured to be within datacenters bandwidth capacity using (4). Equation (5) is intended to ensure sufficient numbers of VMs of the required types in each datacenter. Servers compute capacities are enforced using (6). Equation (7) sums up the cost associated with each datacenter, which is composed of two components: intra-rack cost, given by (8), and inter-rack component, given by (9). Equation (10) ensures the traffic processing capacity of each VM is not exceeded. Equations (11) and (12) tie the variables for number of VMs (i.e., $n^{d,s}_{a,i}$) and traffic (i.e., $q_{d,a,i,vm,s,i',vm',s',l}$) to each other. Flow conservation of nodes is guaranteed by (13). Inequality (14) ensures no ISP backbone link gets congested (i.e., by getting a traffic volume of more than a fixed fraction β of its maximum capacity), while $Path_{e \to d}$ is a path from a precomputed set of paths from e to d. The ILP decision variables are shown in (15).

B DSP and SSP Algorithms

As described in §4.3, due to the impractically long time needed to solve the ILP formulation, we design the DSP and SSP heuristics for resource management. The ISP global controller solves the DSP problem to assign suspicious incoming traffic to data centers. Then each local controller solves an SSP problem to assign servers to

VMs. Figure 16 and 17 show the detailed pseudocode for the DSP and SSP heuristics, respectively.

```
▷ Inputs: $L$, $T$, $DAG_a^{annotated}$, $C_d^{link}$, and $C_d^{compute}$
▷ Outputs: $DAG_{a,d}^{physical}$ and $f_{e,a,d}$ values

Build max-heap $T^{maxHeap}$ of attack volumes $T$
while $!Empty(T^{maxHeap})$
    do $t \leftarrow ExtractMax(T^{maxHeap})$
        $d \leftarrow$ datacenter with min. $L_{t.e,t.d}$ and cap.$> 0$
        ▷ enforcing datacenter link capacity
        $t_1 \leftarrow min(t, C_d^{link})$
        ▷ compute capacity of $d$ for traffic type $a$
        $t_2 \leftarrow \frac{C_d^{Compute}}{\sum_i \frac{\sum_{i'} W_{a,i' \to i}}{P_{a,i}}}$
        ▷ enforcing datacenter compute capacity
        $t_{assigned} \leftarrow min(t_1, t_2)$
        $f_{e,a,d} \leftarrow \frac{t_{assigned}}{T_{t.e,t.a}}$
        for each module type $i$
            do ▷ update $n_{a,i}^d$ given new assignment
                $n_{a,i}^d = n_{a,i}^d + t_{assigned}^d \frac{\sum_{i'} W_{a,i' \to i}}{P_{a,i}}$
        $C_d^{link} \leftarrow C_d^{link} - t_{assigned}$
        $C_d^{compute} \leftarrow C_d^{compute} - t_{assigned} \sum_i \frac{\sum_{i'} W_{a,i' \to i}}{P_{a,i}}$
        ▷ leftover traffic
        $t_{unassigned} = t - t_{assigned}$
        if ($t_{unassigned} > 0$)
            then $Insert(T^{maxHeap}, t_{unassigned})$
for each datacenter $d$ and attack type $a$
    do Given $n_{a,i}^d$ and $DAG_a^{annotated}$, compute $DAG_{a,d}^{physical}$
```

Figure 16: Heuristic for datacenter selection problem (DSP).

```
▷ Inputs: $DAG_{a,d}^{physical}$, $IntraUnitCost$, $InterUnitCost$,
and $C_{d,s}^{compute}$ values
▷ Outputs: $n_{a,i}^{d,s}$ values

while entire $DAG_{a,d}^{physical}$ is not assigned to $d$'s servers
    do $N \leftarrow v_{a,i}^{annotated}$ whose all predecessors are assigned
        if ($N == NIL$)
            then $N \leftarrow v_a^{annotated}$ with max $P_{a,i}$
        $localize$(nodes of $DAG_{a,d}^{physical}$ corresponding to $N$)

▷ function $localize$ tries to assign all of its
input physical nodes to the same server or rack
$localize$(inNodes){
assign all inNodes to emptiest server
if failed
    then assign all inNodes to emptiest rack
        if failed
            then split inNodes $V_a^{physical}$ across racks
update $n_{a,i}^{d,s}$ values
}
```

Figure 17: Heuristic for server selection problem (SSP) at datacenter d.