Enhancing Security Management at Software-Defined Exchange Points

Himal Kumar, Hassan Habibi Gharakheili, Craig Russell, and Vijay Sivaraman, *Member, IEEE*

***Abstract*—Distributed Denial-of-Service (DDoS) attacks continue to escalate in size and scale, and there is growing need for security management at network-level that can restrict a service to a geography (aka geo-blocking) and prevent the victim's IP address from being faked (aka IP-spoof protection). The former reduces the attack surface on the victim, while the latter reduces liability on the organization from which the attack originates. Unfortunately, these solutions are hard to implement in today's networks, requiring expensive hardware appliances and/or manual configuration. This was exemplified in the recent attack on the Australian government census website, which had to be brought down for weeks in order for security configurations to be applied.**

In this paper, we first argue that an Internet Exchange Point (IXP) is an appropriate place for managing security of an enterprise, and then design, implement, and evaluate a geo-blocking and IP-spoofing protection solution for a Software Defined IXP. Our first contribution is to define a grammar for operators to specify their high-level security intents, and a compiler that automatically synthesizes these to low-level flow rules for insertion to the interconnect fabric. Our second contribution is to develop a mixed integer linear program optimization framework for distributing flow rules across switches with limited table size, while minimizing carriage costs of malicious and extraneous traffic. Finally, we evaluate the cost benefits of our scheme via simulation of a large IXP network, and demonstrate its practical utility in blocking attacks via implementation over the open-source ONOS controller and experimentation in an SDN testbed.

***Index Terms*—Security Management, Software-Defined Exchange Point, Geo-Blocking.**

I. Introduction

DISTRIBUTED Denial-of-Service (DDoS) attacks are growing at an alarming rate of 60% by volume and 150% by frequency annually [2], and 2017 saw news web-sites (Al Jazeera, Le Monde), government web-sites (US FCC, Australian ABS), and financial institutions (Bitcoin exchange, South Korean banks) attacked among many others. Our particular motivation for this work comes from the meltdown of the Australian Census web-site in August 2016 [3], which is believed to have been caused by large-scale off-shore DDoS attacks, leaving millions of Australians unable to take the census and causing national embarrassment for the Australian Bureau of Statistics (ABS). When the ABS in response sought to geo-block site-access to Australians, the complexities of manual configuration (black-listing and white-listing of IP address blocks) became apparent, with necessary support services (like DNS) failing and the site being down for several more days. This paper aims to demonstrate that such security attacks can be managed in an automated and error-free manner using the powerful paradigm of Software Defined Networking (SDN).

We focus specifically on two requirements of security management: geo-blocking and IP-address spoofing. We believe that the combination of these two provides an effective (though not comprehensive) way of protecting critical web-services. *Geo-blocking* refers to the ability to restrict access to the service (*e.g.,* website) to a geographical boundary, such as a set of countries. Typically, the country codes are converted to a set of Autonomous System (AS) numbers, which are then mapped to a set of IP prefixes. These prefixes are then configured into the enterprise's border router access control lists (ACLs) or border firewall rules (*e.g.,* PacketViper [4]), so that packets arriving from these prefixes are allowed/dropped. These configurations are manual, slow, and error-prone, since a single country may have anything from hundreds to thousands of prefixes. Further, care has to be taken to not disable essential services (like DNS) that may need to use mirrors and caches in different geographical regions in order to function properly. The complexity of configuration, coupled with the expense of border routers and firewalls with appropriate capability, deters many organizations (including governments) from using geo-blocking to protect their services. Even when configured, geo-blocking can be defeated by an attacker who hides their source IP address by spoofing or use of VPN (proxy) services. Therefore, mitigating *IP address spoofing* is important while masking IP addresses by VPN and proxy may not be prevented.

Indeed, an organization (*e.g.,* a University with high-bandwidth network connectivity) can (unknowingly) become the source of a DDoS attack, and it is important for them to block such spoofed traffic as they may otherwise be held liable for the damages caused. Again, current practice requires manual configuration of ACLs on the egress border router to detect and drop packets with spurious source IP address, something that enterprises seldom do, since it requires time and skills, in addition to expense and performance penalties on their border device.

We argue that the geo-blocking and IP-spoof security management capabilities are best offered by the enterprise's peering provider. This will typically be at an Internet Exchange Point (IXP), which might be operated either by a transit provider (like AT&T or Telstra) or an inter-connect provider (like Equinix or Megaport). The reasons for having these

H. Kumar, H. Habibi Gharakheili and V. Sivaraman are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, 2052 Australia. (e-mails: himal.kumar@unsw.edu.au, h.habibi@unsw.edu.au, vijay@unsw.edu.au)

C. Russell is with CSIRO Data61, Sydney, NSW 2015, Australia (e-mail: craig.russell@data61.csiro.au).

This submission is an extended and improved version of our paper presented at the EuroSys 2017 XDOMO Workshop [1].

security services hosted at the peering exchange rather than the enterprise premises are two-fold: (a) IXPs can drop attack traffic *before* it hits the enterprise border device, thereby protecting the (expensive) peering link from volumetric attacks that can cripple availability of the enterprise's services; and (b) the IXP has greater skills and resources to perform and manage configurations to prevent attacks, since their equipment and effort is amortized across their many enterprise customers; further, since the IXP carries both the data-plane packets and control-plane messages for the enterprise, it has requisite visibility needed to "scrub" the traffic in its fabric in-line with the BGP (Border Gateway Protocol) prefixes relevant to the enterprise. Also, IXPs have the opportunity to monetize their infrastructure by offering security management as a value-add service where they can charge the customer enterprise for providing geo-blocking and IP-spoofing services. The pricing model is left to the individual IXP, and can be dynamic depending on policies, number of rules, etc.

We further argue that Software Defined Networking (SDN) is the ideal paradigm for the realization of the above security capabilities. It allows the operator to specify security requirements in terms of high-level policies, which can then be automatically synthesized into low-level flow rules on switches, thereby reducing provisioning time; it allows data-plane forwarding rules to be made consistent with control plane advertisements, thereby reducing errors; and it enables the modular development of security capabilities on rapidly maturing open-source platforms such as ONOS [5], thereby allowing operators to avail of community support.

Our objective in this paper is to design, optimize, and evaluate an SDN-based geo-blocking and IP-spoofing protection solution that IXPs can offer to enterprises. We outline our system architecture, and make three specific contributions. Our **first** contribution is to define a grammar that allows operators to specify the policies in a simple and abstract way, and a compiler that synthesizes these policies into flow-rules, using live BGP information and online geo-mapping services. Since each policy can compile into thousands of flow-rules, our **second** contribution determines the best placement of rules across switches in the distributed fabric, such that carriage of unwanted traffic is minimized while staying within the table size limitations at each switch. For our **third** contribution, we evaluate the efficacy of our rule placement algorithm via simulation on a small illustrative topology and a large real IXP topology, and prototype our solution (specifying high-level security intents, synthesizing and inserting low-level flow rules without optimization) on the open-source ONOS-based SDN interconnect platform and demonstrate its ability to block attacks in a laboratory testbed.

Our solution builds upon our preliminary work in [1] by developing rules placement optimization algorithm and evaluating via simulation on real IXP topology under various scenarios, and to the best of our knowledge is the only geo-blocking and IP-spoof protection solution for IXPs built upon SDN technology. We emphasize that this work is limited in scope to geo-blocking and IP-spoof protection, and does not as such cover all types of DDoS attacks. A web-site that needs to be accessible to all customers across the world cannot be geo-fenced, and hence may not benefit from our solution. However, there are many instances where services do need to be restricted geographically, such as the example of the ABS census in Australia discussed earlier. Further, IP-spoofing remains a major source of DDoS attacks, as stated in the Global Threat Landscape report by Arbor Networks [6]. Our solution provides a baseline protection at Internet Exchange Points against IP-address spoofing by enterprise networks, greatly reducing the attack surface – note that protection against insider attacks, sourced from the same enterprise or domain, in beyond the scope of this work since their traffic does not traverse Internet Exchange Points (to prevent attackers inside the same country/area, our solution needs them to be on a different AS and peer at an exchange point). The focus of our work is on showing the feasibility of implementing the protection mechanisms at an IXP using SDN techniques; further, this is done in a scalable way with low cost of rules placement in switches, also consistent with prior empirical studies [7] showing that the geo-spatial distribution of sources for recent DDoS attacks is often limited to a set of prefixes rather than being random across the globe.

The rest of this paper is organized as follows: §II describes prior related work. In §III we describe our system architecture and develop the policy grammar and compiler, while optimal placement of flow rules amongst switches is studied in §IV. We implement and evaluate our scheme in §V, and the paper is concluded in §VI.

II. Related Work

In this section we briefly review prior works in the areas of geo-blocking / IP spoof-protection, SDN-based IXP architectures, SDN policy intent frameworks, and SDN rule-placement that are relevant to our work.

Geo-blocking, where deployed today, is largely based on commercial solutions such as Packet Viper [4] and Next-Generation Firewall (NGFW) [8]–[10]; these are however custom-hardware solutions that tend to be very expensive and are hence outside the reach of most enterprises. Further, geo-filtering may need to be tuned at run-time depending on the attack patterns [11], which is not always easy with embedded hardware solutions. IP source address spoof protection is recommended best-practice by the IETF in the form of Network Ingress Filtering [12], but is highly manual and is again not implemented by most enterprise network operators.

SDN-based IXP architectures have become an area of active research in recent years – Software Defined eXchanges (SDXes) [13], [14] have been proposed as a means of overcoming the shortcomings of BGP in order to do better traffic engineering and load balancing. Work in [13] aims to optimize composition of policies from participants (enterprises) connected to an IXP by compressing the forwarding table of switching fabric. Authors further enhanced their method of reducing both forwarding table size and the time to compute these entries by several orders of magnitude in [14]. SDN-based architectures were deployed in production IXPs in New Zealand in 2013 [15] and France in 2015 [16], demonstrating feasibility and innovative capabilities not possible before for IXP operators as well as customers. However, none of them

optimize the number of rules installed in the switching fabric. Recently, Google revealed their SDN-based edge peering architecture [17] that enables application-aware routing using fine grained traffic engineering capabilities, and currently serves 22% of Google's total traffic to Internet. In our previous work we have built an OpenFlow-based interconnect called CaSToR [18] that is now part of the standard release of the ONOS open-source SDN controller. While our previous work focused on more flexible interconnects for enterprises to seamlessly peer with cloud providers across public and private fabrics, the focus of the current work is on security. Since IXPs are a central entity where large amounts of Internet traffic are exchanged across a large number of domains, they are an ideal location for implementing security solutions including black-holing [19]–[21].

The maturation of SDN has also witnessed a rise in frameworks that allow network policy to be specified at a high level. Intent Frameworks [22]–[26] have been proposed to allow specification of high-level policies related to connectivity, ACLs, YANG models, etc., which are then compiled down to low level network configurations – this automation reduces manual complexity and human errors. A recent framework called Propane [27] provides constructs to automate BGP configurations in network devices using high level abstract language.

Rule placement in SDNs has been studied under various contexts. iSDX [14] aims to overcome the limited capacity of flow tables in Openflow devices by implementing multi-table pipelining and packet encoding to compress and reduce the number of rules requiring installation. The authors in [28] talk about trading routing paths with limited flow space, and distribute the rules in a separate path with flow space availability – the resulting changes in routing paths may however not always be feasible or desirable. Difane [29] uses integer linear programming to optimize the rule placement by classifying rules and caching the most important ones at authority devices, thereby reducing the load on the controller and the number of rules required. Palette [30] decomposes large SDN tables into smaller ones and distributes them across the network, while OneBigSwitch [31] uses multiple paths along with routing policy to produce efficient rules placement that distribute forwarding policies across SDN networks.

The most closely related works are [13], [14] which primarily focus on connectivity rules while this paper focuses on blocking rules for security management. Also, these prior works optimize the number of SDN rules pushed into switches to avoid explosion of flow-tables, whereas our scheme optimizes the placement of rules to reduce the cost of carrying malicious traffic. We believe that our solution complements these relevant proposals where they compress rules and our method optimizes the placement of rules.

III. System Architecture and Policy Grammar

A. System Architecture

The high-level architecture of our geo-blocking and IP-spoof protection solution at the IXP is shown in Fig. 1.

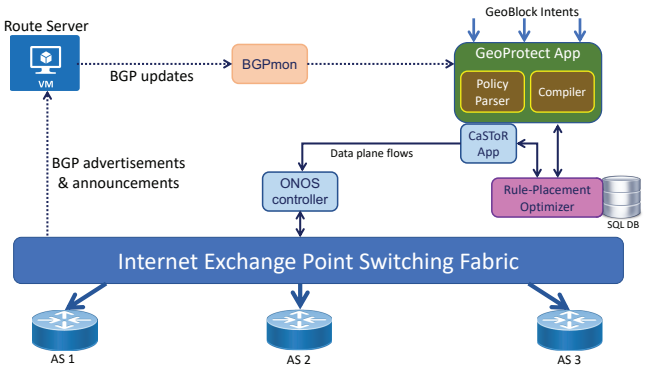


Fig. 1. GeoProtect architecture including geo-blocking and IP spoof-protection.

The base IXP platform consists of the data-plane, which is a switching fabric comprising (a potentially large number of) Layer-2 switches, and a control-plane, which comprises (a redundant set of) route-servers that perform BGP peering with the various inter-connecting domains in order to redistribute the routes. Our implementation (detailed in V-C) uses the NoviFlow NS1132 switches with full OpenFlow 1.3 support in the data-plane, and the open-source SDN controller ONOS [5] in the control-plane; we note that ONOS is carrier-grade with full redundancy, provides an intent framework [23] for topology abstraction, and is well-supported in the community. Further, we use the CaSToR inter-connect application we have developed in previous work [32], which allows the operator to provision and manage inter-connect customers, sets up the BGP sessions in the control-plane, and creates the inter-connectivity rules in the data-plane. We note that our CaSToR SDN application is fully open-source, and is bundled into the standard ONOS release. We have deployed our CaSToR platform in two production exchange networks namely Amlight in Florida, and CENIC in California. The operational trial of our CaSToR was demonstrated at International GLIF workshop with participants of National Research and Education Network (NREN) operators [33].

The "GeoProtect" app (shown as a green box in Fig. 1) developed in this paper provides the geo-blocking and IP-spoof protection capabilities. It takes policies from the operator in the form of "intents", shown by arrows in the right-top of the figure. The grammar used for these policies, and their automated parsing and compilation into ONOS intents and switch flow-rules, will be described next. The compilation process requires the GeoProtect app to access a geo-location database (we use the Maxmind [34] and Hurricane [35] services), as well as a feed of the BGP udpates in order to know the IP prefixes belonging to each connected organization (we use BGPmon [36] that peers with the route server and outputs BGP updates in easy to parse XML format). The GeoProtect app locally caches this information in an SQLite database, and passes the resulting flow-rules on to CaSToR for insertion into the data-plane.

The flow-rules to be inserted into the data-plane can be large in number, and since the SDN switches have limited table-size, it may in general not be possible to fit all rules into the first switch along the path of the traffic. This entails a penalty since traffic may be unnecessarily carried across switches only

to be dropped later. In §IV we develop a method to optimize rule-placement (pink box in Fig. 1) so as to minimize this penalty subject to flow-table size limitations. A comprehensive evaluation of our scheme, both via simulation of small/large IXP networks and a full implementation with lab trials, will be presented in §V.

B. Policy Grammar

In order to reduce provisioning time and human error, we envisage that geo-blocking and IP-spoof protection policies will be specified by the operator in an abstract and succinct manner, and these will be automatically synthesized into the large number of flow-rules required at network switches. In this section we define the high-level grammar for the policies, and design a compiler that dynamically converts these to flow-rules by leveraging openly available IP prefix geo-location databases along with the enterprise's own BGP advertisements.

1) Geo-Blocking: Geo-blocking allows a service to be geographically restricted. A geo-block policy is defined with the following grammar:

def geoblock (**Policy** Id) {
  Source = [**Country** (Codes), **AS** (AS no.),
    IP (IP prefixes)]
  Destination = [**Domain** (*e.g.,* `www.example.com`),
    AS (AS no.), **IP** (IP prefixes)]
  Classifier = [**Port** (*e.g.,* 80), **Protocol** (*e.g.,* tcp)]
  Exceptions = [*e.g.,* DNS, `amazon.com`, `google.com`]
  Time = []
  Action = ALLOW / BLOCK }

The *def* keyword is used to define the policy with a unique associated ID. The policy contains the following attributes:

- *Source*: can be a combination of standard country codes (*e.g.,* US, UK, AU), AS numbers, and IP prefixes, all separated by commas. They represent the places from where the traffic is originating.
- *Destination*: refers to the domain, AS number, or IP prefixes of the services to which the traffic is destined. This is typically whole or a part of the network of the enterprise seeking geo-fencing for its service.
- *Classifier*: (optional) restrictions on source or destination space, such as specific port numbers or protocols (*e.g.,* port 80 web-traffic).
- *Exceptions*: (optional) essential services, domains, or areas which should not be blocked. Examples include global DNS servers, and trusted cloud services like Amazon or Microsoft Cloud that interact with the enterprise service (*e.g.,* for analytics). Specific IP prefixes which are being used to manage or control the service should also be enumerated here.
- *Time*: (optional) duration for which the geo-block policy should remain active; defaults to infinite if not specified, and will remain in effect till the policy is manually withdrawn.
- *Action*: specifies the action for traffic matching the source, destination, and classifiers specified in the policy. Specifying action ALLOW makes this a white-list policy (*i.e.,* connectivity rules are created), whereas action BLOCK makes this a black-list policy (*i.e.,* blocking rules are created).

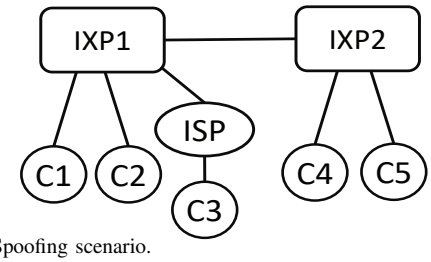


Fig. 2. IP Spoofing scenario.

Referring back to the example of the Australian census site, a simple geo-block policy that restricts access to the site to Australians can be specified using the above grammar as: {source=AU, destination=`www.abs.gov.au`, exceptions = DNS, action = ALLOW}. As another example, a video streaming entity (*e.g.,* Netflix) that does not want its service to be reachable from Country "X" (for legal or compliance reasons) can specify a policy {source=X, destination=`www.netflix.com`, action = BLOCK}. We will soon see how this policy gets translated to flow-rules by leveraging existing geo-location databases.

2) IP-Spoofing Protection: Geo-blocking relies on the source IP address in a packet being correct. An attacker can circumvent geo-blocking by spoofing their source IP address. For example, consider Fig. 2 in which customers C1 and C2 are in the same country, inter-connecting at IXP1, while C3 is in a different country, and connects to the IXP1 via a transit ISP. If C1 has geo-blocked its service to its own country, an attacker in C3 can spoof the address of C2 to circumvent the geo-block and directly attack C1's service. Equally concerning, an attacker in C4 (connected to another IXP2) can also launch an indirect attack on C1 by spoofing C1's address to send request traffic to a service such as DNS that can amplify and reflect the response attack back on to C1. Spoofing can therefore not only bypass security measures and enable reflection attacks, but also create liability for the entity whose IP address was illegitimately used.

The two IXPs in the above example are well positioned to block IP spoofing, by ensuring that packets entering their fabric from a connected entity carry a source IP address that is part of the prefixes advertised (in its BGP messages) by the connected organization. This is a particularly cost-effective way of dropping spoofed packets from stub ASes (as opposed to multi-homed transit ASes), and is hence suitable for most enterprises.

An operator can specify an IP-spoofing protection policy using the following grammar:

def spoof_protect (**Policy** Id) {
  Customer = [IP address of the Border Router] }

The policy above requires only one parameter – the IP address of the connected entity's Border Router. This allows the IXP to use the BGP advertisements from that router to dynamically deduce the legitimate source IP addresses for that entity, and is robust to changes in IP addresses over time. Ensuring the authenticity of BGP advertisements [37] is beyond the scope of this paper.

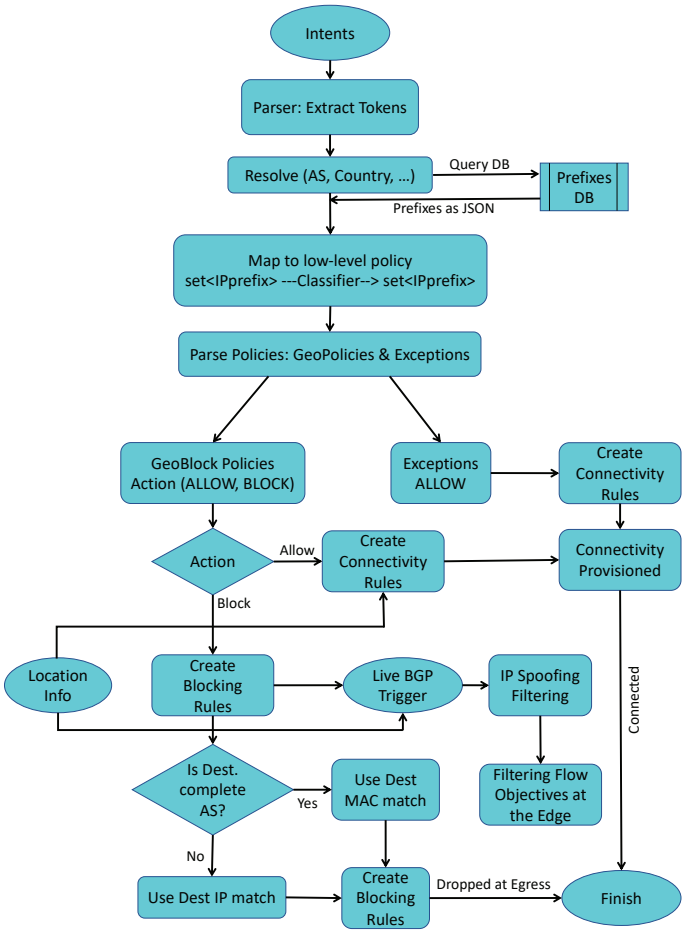


Fig. 3. Compiler algorithm.

C. Compilation

We now design and implement a compiler that translates the high-level geo-blocking and IP-spoof protection "intents" above into low-level network flow rules which can be installed into the switching fabric. Fig. 3 shows a detailed flowchart of the compilation process of the geo-blocking rules, and relies on interaction with: (a) a geo-mapping database that associates country codes, AS numbers, and IP prefixes; (b) an SDN controller (ONOS is our case) that converts connectivity intents into topology-specific flow objectives; and (c) an SDN inter-connect application (CaSTOR in our case) that manages basic connectivity at the IXP.

In the first step, a *Policy Parser* receives the intents and parses them into useful tokens; we use the ANTLR [38] tool that maps tokens to JAVA classes and objects. The *source* and *exception* arguments are extracted, and used to query the Maxmind [34] geographical IP prefix ownership database via REST APIs to obtain their corresponding address blocks. Lower-level policies are formulated using the IP prefixes which contains the set of sources, destinations and network classifiers. From here, exceptions are separated from the policies and are treated separately. Exceptions are always allowed. Therefore, they are converted into ONOS connectivity intents and are ultimately translated to network flow rules.

If the default action was ALLOW, connectivity flow rules are created to allow traffic from the specified sources to the specified destinations that match the specified (if any) classifier. If the default action was BLOCK, the sources specified needs to be blocked from accessing the specified destinations, and appropriate blocking rules need to be inserted in the data-plane fabric. In case of conflicting policies (one specifying ALLOW and another one specifying BLOCK for a given prefix), BLOCK overrides ALLOW. The appropriate placement of these blocking rules in the fabric is the subject of the next section Information about the location of the sources and destinations is provided by CASToR based on customer provisioning and BGP prefix exchanges. A suitable destination match for the flow-rules is selected based on whether the destination is a subset of customer's network, in which case IP prefixes supplied in the destination are used as a match, or if the destination is the entire customer network, in which case a smaller number of flow rules are required that match on customer destination MAC address. Source IP filtering rules for spoofing protection are created using the BGP information received through BGPMon. These IP spoofing filtering rules are regularly updated whenever there is a new BGP update which is shown in the flow chart as "Live BGP Trigger".

IV. Optimizing the Blocking Rules

As discussed in the previous section, geo-blocking and IP-spoofing protection policies result in three types of Openflow rules: 1) Connectivity rules – these are created to allow the traffic from certain prefixes and are installed along the path. The position of these rules is fixed, and strictly depends on traffic paths. 2) IP spoofing protection filtering rules – these rules filter traffic entering the exchange and are only placed at the edge nodes where a customer connects to the fabric. 3) Blocking rules – these rules are created to block the traffic originating from certain prefixes and can be placed anywhere on the path to drop the traffic. Connectivity rules are required to be placed along the path of the flow, while filtering rules are required to be placed at the switch where the customer traffic enters the IXP fabric. Therefore, blocking rules (which can number in the thousands) are the only ones for which the IXP operator has the freedom to move amongst the switches in the exchange fabric so as to drop the traffic wherever feasible along the path.

The best possible placement of blocking rules is at the edge, as close as possible to the source, blocking malicious traffic before entering the exchange fabric. In the event that the edge switch does not have enough flow-table space, overflow rules can be placed at the next switch along the flow path. A "greedy" approach for placing blocking rules might therefore first sort all flows in descending order of volume, and place them one-by-one at the first node along the path where space is available. However, as we will show, the greedy approach is short-sighted since it does not have a global view that considers correlations amongst flow paths. In what follows we formulate the placement of blocking flow rules as a mixed integer linear program (MILP) that can be solved to obtain a globally optimum solution. We first describe the factors that affect rule placement:

- **Path**: Each rule is associated with a definite path in the fabric starting from the source peer (*i.e.,* the closest IXP

customer originating the traffic) to the destination peer (IXP customer who is passing the geo-block policy). A rule can only be placed at a node which happens to be on the path associated with it. The path is the route which will be taken by the traffic being matched by this rule if it were not to be dropped or blocked.

- **Volume**: Each flow matching a specific rule carries a certain volume of traffic, which unnecessarily consumes network bandwidth as it traverses each link in the exchange fabric. It is therefore sensible to prioritize higher volume flows over lower volume flows so they can be dropped earlier along the path. Since traffic rates can be dynamic, our method keeps track of average flow volume over an epoch (or pre-defined duration), and will re-optimize the placement of rules at the end of each epoch.
- **Flow Table Space**: This corresponds to the number of entries available in the flow table to accommodate blocking rules at each node or switch. This is hardware-specific and can be in the hundreds or thousands, and is further shared between connectivity rules and filtering rules of our application, along with rules from other possible SDN applications.

We mathematically model the impact of the above inputs on rule placement, so as to minimize the carriage of unwanted traffic in the IXP fabric.

A. Problem Formulation

Our objective is to optimize the placement of flow rules at IXP switches, given network topology and flow rates. Let R denote the blocking rules to be placed after the geo-blocking compilation, each rule corresponding to a single flow in the switching fabric matching on a source IP prefix. The rule set $\mathcal{R}$ is denoted by:

$$\mathcal{R} = \{r_1, r_2, r_3, ..., r_R\} \tag{1}$$

Let $\mathcal{F}$ be the set of average traffic volumes matched by each rule in a time period. This is measured in our CaSToR platform by harvesting flow table statistics from the switches in real-time.

$$\mathcal{F} = \{f_1, f_2, f_3, ..., f_R\} \tag{2}$$

Let the number of nodes/switches in the IXP network be denoted by N, and the set of nodes is denoted by $\mathcal{V}$, where:

$$\mathcal{V} = \{v_1, v_2, v_3, ...v_N\} \tag{3}$$

The table size at a node j, representing the available flow table capacity of each node, is given by T_j . Note that this is the available flow table space for blocking rules only, and excludes the flow table space used up by other rules (*e.g.,* connectivity rules and rules from other SDN applications).

Let a path $\mathcal{P}$ corresponding to a rule be defined as a sequence of given nodes belonging to $\mathcal{V}$, and is written as:

$$\mathcal{P} = \langle v_n \ \mid v_n \in \mathcal{V} \rangle \tag{4}$$

An example path P_i of a flow corresponding to rule r_i can be written as a vector:

$$P_i = \langle v_1, v_5, v_4 \rangle \tag{5}$$

We define hop-length h_{ij} as the number of hops to node v_j on the path of the flow associated with rule r_i. Following on from the example path in (5) above, the hop-lengths for this flow are: $h_{i1} = 0$, $h_{i5} = 1$, and $h_{i4} = 2$. The hop-length is related to the cost of placing a rule at a node on the path of a flow, since placing the blocking rule at a node with higher hop-length requires carrying the flow traffic over a larger number of hops. The cost of placing the rule at the first node is zero, as the traffic does not use any link capacity inside the exchange fabric, and the cost increases by one with every hop on the path. In this case, the cost of placing the rule at node v_4 is 2 as it is the third node on the path and the blocked traffic has traversed two links on the path.

The optimization is performed over the variables x_{ij}, which indicate whether or not the rule r_i is placed at node v_j, namely:

$$x_{ij} \ = \begin{cases} 1 & : \text{if rule } r_i \text{ is at node } v_j \\ 0 & : \text{otherwise} \end{cases} \tag{6}$$

The cost of placement of rules can now be obtained by summing over all rules the flow volume for that rule multiplied by the hop-length of the rule along the flow path:

$$C = \sum_{i=1}^{R} \left[\sum_{v_j \in P_i} x_{ij} * h_{ij} \right] * f_i \tag{7}$$

This equation defines the objective function of our optimization problem. Minimizing this cost function yields values of x_{ij} which specifies the node v_j where rule r_i should be placed. The optimization has to account for the following constraints: For a given rule, it should only be present at one node on the path:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad \forall i \in \Big\{1, 2, \ldots, R\Big\} \tag{8}$$

The number of rules placed at a node v_j should not exceed the flow-table capacity T_j at node v_j, *i.e.,* :

$$\sum_{i} x_{ij} \leq T_j, \quad \forall j \in \Big\{1, 2, \ldots, N\Big\} \tag{9}$$

Finally, x_{ij} can have a non-zero value only if node v_j is on the path of the flow corresponding to rule r_i:

$$x_{ij} = 0, \quad \forall j \notin P_i \ \text{ and } \ i \in \Big\{1, 2, \ldots, R\Big\} \tag{10}$$

B. Solving the optimization and interpreting the results

We now convert our optimization formulation into a Mixed Integer Linear Program (MILP) form, so we can solve it using MATLAB. A general MILP problem structure minimizes an objective function subject to a series of equality, inequality, and upper/lower bound constraints. We are easily able to map our problem to the MILP structure, with (8) and (9) becoming the equality and inequality constraints respectively, the variables x_{ij} lower and upper bounded by 0 when node v_j

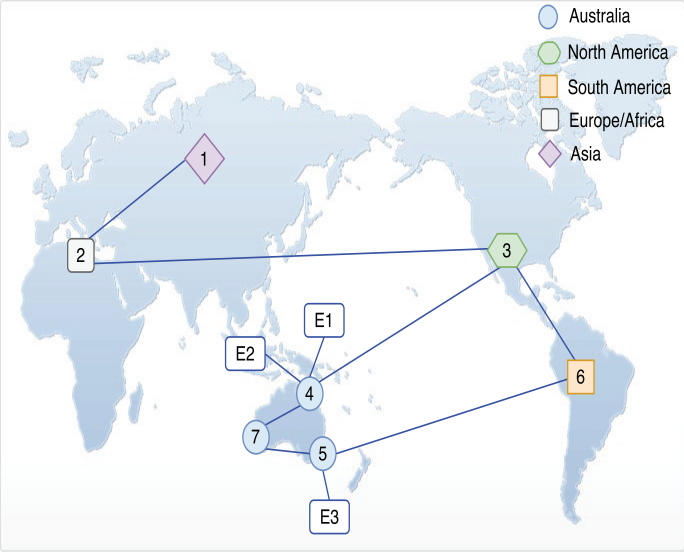


Fig. 4. Experimental Exchange topology.

is not in the path for flow corresponding to rule r_i as per (10), and lower and upper bounded by 0 and 1 otherwise, and the objective function used as is. Solving the MILP optimization yields optimal node of placement for each rule. We now talk about how the optimal solution is interpreted by our geo-block application next.

The optimization algorithm is run periodically. This period is a configurable parameter, the shorter the more beneficial to network management. Note that the practical value depends on two factors namely, the speed of computing and the capability of switches in handling frequent control messages – we chose an "epoch" of 15 minutes as it is a reasonable time-period over which traffic patterns vary. This is also consistent with network management practice that uses tools like MRTG to poll interface counters every 5 minutes and making judgments using three successive values. However, we choose to reconfigure the placement of rules only if the gains in minimizing carriage of unwanted traffic outweighs the cost of control message overheads needed for reconfiguration. We therefore define a tunable decision parameter, called "benefit" that is the ratio of the cost reduction to the number of rule replacements required:

$$\text{Benefit} = \frac{\text{Cost}_{CP,\ NV} - \text{Cost}_{NP,\ NV}}{\text{Number of replacements}} \tag{11}$$

where NP denotes the New Placement of rules, CP denotes the Current Placement and NV represents the New Volume measured in the current epoch. The higher the value of benefit, the more is the cost gain and the more likely we would like to reconfigure our devices. The determination of the threshold value of benefit has to be tuned to the size of the network, number of controller nodes and the capability of devices to handle OpenFlow updates. We note that benefit is only used when the placement changes due to volume changes and not policy changes. Whenever a new policy is created or a current policy is modified, the calculated optimal placement of rules is always used to place the new and existing rules. The role of the benefit decision parameter is explored next in the experiments section.

TABLE I
NUMBER OF PREFIXES

Country	# Prefixes	Country	# Prefixes
AU	6,384	SG	1,988
US	50,827	FR	14,852
CA	10,280	IT	3,987
CN	6,419	IES	4,062
IN	4,441	NL	8,331
IE	1,832	PL	3,497
JP	4,429	HK	2,968
TR	1,063	BR	2,819
MY	750	ID	1,611

TABLE II
FLOW TABLE CAPACITIES

Node ID	Available capacity
1	4,000
2	1,000
3	20,000
4	3,000
5	30,000
6	20,000
7	5,000

V. EXPERIMENTATION AND EVALUATION

We now evaluate the efficacy of our rule placement algorithm via simulation on a small illustrative topology and a large real IXP topology (in MATLAB environment on a machine with 4 cores of CPU and 16GB of RAM), and prototype our entire solution on the open-source ONOS-based SDN interconnect platform and demonstrate its ability to block attacks in a laboratory testbed.

A. Evaluation in Microscopic Topology

We start with a simplified illustrative topology to demonstrate the intuition behind the placement of policies and their optimal placement across the fabric. We consider three policies over a topology consisting of seven nodes, and show how our optimizer places/moves rules to reduce the carriage cost of unwanted traffic. Our simplified topology is shown in Fig. 4. Various color-coded and numbered shapes placed on the world-map denote IXP nodes aggregating traffic from different geographical areas and are connected together as a single distributed exchange fabric. In this topology, there are three nodes in Australia (*i.e.,* nodes 4, 5 and 7), one in Asia (*i.e.,* node 1), one in Europe/Africa (*i.e.,* node 2), one in North America (*i.e.,* node 3), and one in South America (*i.e.,* node 6). Networks of E1, E2, and E3 represent three enterprises (or IXP customers) located in Australia, and are connected to the exchange fabric at nodes 4 and 5. In our simulation, we do not have access to live BGP advertisements, and hence we use the shortest path to route traffic traversing between nodes.

Let's consider following geo-block policies expressed by E1, E2, and E3:

- P1: **def geoblock** (**Policy** 1) **Source** = [**Country** (AU)] **Destination** = [E1] **Action** = ALLOW
- P2: **def geoblock** (**Policy** 2) **Source** = [**Country** (CN, IN)] **Destination** = [E2] **Action** = BLOCK
- P3: **def geoblock** (**Policy** 3) **Source** = [**Country** (US, CA)] **Destination** = [E3] **Action** = BLOCK

P1 represents a policy where E1 allows only traffic with source address of Australia prefixes which means that con-

nectivity rules need to be created only for Australia. These connectivity rules from Australia are installed at nodes 4, 5, and 7. Enterprise network E2 intends to block traffic sources from China and India that attempt accessing its services, hence expresses the policy P2 – In this topology, E2 represents a streaming content provider who does not offer services to users in China and India. Lastly, enterprise network E3 requests the policy P3 to deny service access from prefixes of US and Canada.

Blocking rules compiled from the policy P2 are associated with the path: 1→2→3→4 (starting at node 1 for China and India, terminating at node 4 for E2 in Australia). Similarly, blocking rules compiled from the policy P3 are associated with the path: 3→6→5 (starting at node 3 for US and Canada, terminating at node 5 for E3 in Australia).

Table I shows the number of IPv4 prefixes for selected countries (AU: 6384; CA: 10280; US: 50827; CN: 6419; IN: 4441) as per Maxmind database [34] during our experiments. For blocking policies (*i.e.,* P2 and P3), every prefix becomes a rule that is installed at a node along the respective paths mentioned above. We note that the number of blocking rules generated for policy P2 and P3 equals to the total number of prefixes in corresponding source countries. Therefore, P2 and P3 collectively translate into 71,967 Openflow rules (*i.e.,* total number of prefixes in CN, IN, US, and CA).

Table II shows the available capacity of flow table at each node to accommodate blocking rules. Based on our simulated policies, traffic paths, and prefixes count we have customized available capacity in each switch to impose restrictions, and hence illustrate the need for optimization. For an optimal placement, we only focus on blocking rules resulted from policies P2 and P3. As discussed earlier in §IV, we can only change the placement of blocking rules – no control is applied to connectivity rules and their placement. Once our geo-protect application translates the policies into Openflow rules, they are fed to our optimizer which then computes the optimal placement of each and every rule considering the capacity of flow tables and the rate of traffic associated with individual flows. Our optimization algorithm runs every fifteen minutes to recompute new placements. In what follows we demonstrate the performance of our optimizer in three scenarios.

1) Scenario 1: Equal Volume: We start with a baseline scenario in which a consistent unit of volume for each rule is used – rules are treated equally (*i.e.,* same weight) and differ only in their corresponding paths. Fig. 5a depicts the output of our optimizer for placing these rules.

Rules are distributed along the respective paths satisfying the flow table capacity constraint. We can see rules placement at each node (shown by x-axis in Fig. 5a) for each country source specified in the blocking policies (*i.e.,* CN, IN, US, CA).

We note that the aggregate traffic rate from each source country is shown in Fig. 5f. In our first scenario (captured by the first fifteen minutes in Fig. 5f), all rules are treated equally thus their rate is equally set to one unit (*i.e.,* 1 Mbps). Rules are placed based on their paths and available table space. Unsurprisingly, rules with source prefixes from China and India are placed only at nodes 1, 2, 3, and 4, and from US and Canada at nodes 3, 6, and 5. For example, according to Fig. 4 traffic from North America (node 3) to E3 in Australia (node 5) traverses the shortest path (node 6) and is blocked by the policy P3. Lastly, the placement cost of this scenario amounts to 83,654 (*i.e.,* the sum of hop-length for individual blocking rules).

TABLE III
BENEFIT OF REPLACEMENTS.

Scenario	# replacements	Cost (NP)	Cost (CP)	Benefit
Scn. 1	-	83,654	-	-
Scn. 2(a)	48,150	$2.68e6$	$3.81e6$	23.63
Scn. 2(b)	22,463	$2.88e6$	$3.38e6$	22.54
Scn. 2(c)	11,354	$2.91e6$	$2.96e6$	4.43
Scn. 3	12,550	$2.82e6$	$3.30e6$	38.75

TABLE IV
RULES PLACEMENT FOR SCENARIO 2(C).

Source	node 1	node 2	node 3	node4	node 5	node 6
CN	524	552	2345	2998	0	0
IN	3476	448	515	2	0	0
US	0	0	6861	0	23967	19999
CA	0	0	10279	0	0	1

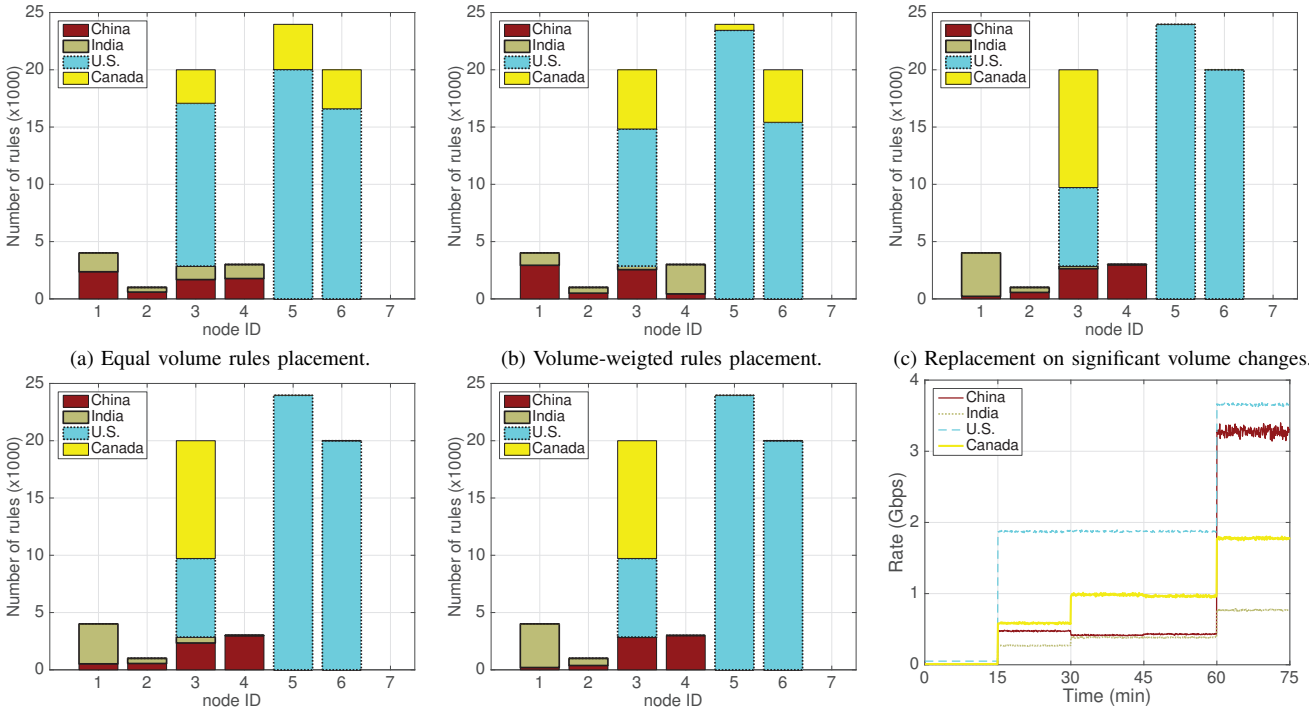
TABLE V
RULES PLACEMENT FOR SCENARIO 3 (DDOS ATTACK).

Source	node 1	node 2	node 3	node4	node 5	node 6
CN	200	376	2843	3000	0	0
IN	3800	624	17	0	0	0
US	0	0	6862	0	23967	19998
CA	0	0	10278	0	0	2

2) Scenario 2: Volume Weighted: We now look at the output of our optimizer when rules are weighted by their average volume over the last epoch (*i.e.,* 15 minutes in our experiments) – the placement is re-optimized every fifteen minutes. Note that we use an exponential random variable for the volume of flows with varying mean values ranging from 20 to 80 Mbps. In this scenario, the aggregate rate of traffic from each country is captured by three slots of fifteen minutes each, between [15, 60] min in Fig. 5f. Now, every rule is weighted by the average volume of traffic seen in the past fifteen minutes. Our optimizer prioritizes high volume rules by placing them closer to their source, thus reducing the overall cost.

Fig. 5b shows the rules placement computed by the optimizer. We can see that there are significant changes in the distribution of rules at all nodes compared to of scenario 1 in Fig. 5a. We observe that more than 50% of rules from India (*i.e.,* 2,564 out of 4,441) are placed further down on the path at node 4, as they carry less volume of traffic compared to those from China. On the other hand, it is seen that majority of large volume rules associated with Canada are placed at nodes 3 and 6 (5171 and 4586 rules respectively) meaning closer to their traffic source, and due to flow table capacity constraint only 5% of rules (*i.e.,* 523 rules) are placed at node 5.

Row Scenario 2(a) in Table III shows the number of rules replacements required to reach this state from previous epoch (*i.e.,* the baseline) and the associated benefit. For our experimental topology, we see a high benefit of 23.63 (with 48150 replacements) and therefore the GeoProtect application accepts the optimizer's output and reconfigures devices with



(a) Equal volume rules placement.
(b) Volume-weigted rules placement.
(c) Replacement on significant volume changes.
(d) Replacement on minor volume changes.
(e) Replacement on major rate changes for selected DDoS sources.
(f) Variation of traffic volume.

Fig. 5. Placement of rules and traffic rates from each country.

the new placement of rules.

Over the next epoch (*i.e.,* [30, 45] min; Scenario 2(b)), we change the traffic volume for flows of Canada, China, and India while keeping the U.S. traffic fairly consistent compared to the previous epoch. Fig. 5c depicts the new placement of rules computed by the optimizer. Since the volume of traffic from Canada has increased, all of its 10,280 rules are pushed higher up along the path towards the source at node 3, while pushing down the U.S. rules (those that have lower rates compared to rules from Canada). We note in Fig. 5f that the aggregate rate of U.S. traffic is higher than of Canada due to higher number of prefixes (*i.e.,* US: ∼51K and CA: ∼10K), but individual rules may have higher or lower rates. Also, due to a decrease in China-sourced traffic and an increase in India-sourced traffic, we observe that rules for India are pushed higher on the path with 3,779 out of 4,441 rules are placed just at node 1. The benefit of replacement for this situation is 22.54 (as shown in the third row of Table III) which is again high enough for the GeoProtect app to reconfigure the network with new optimal placement.

In the last epoch of this scenario (*i.e.,* [45, 60] min; Scenario 2(c)), we keep traffic volume from all sources fairly consistent with the previous epoch – no significant change in traffic volumes, except slight variations in flows sourced from China and Canada as shown in Fig. 5f. The optimizer computes a new placement for this epoch as depicted in Fig. 5d and Table IV. Unsurprisingly, we do not see a noticeable change in the bar graph of Fig. 5d compared to of Fig. 5c in scenario 2(b). We note that the optimizer suggests 11,354 replacements due to slight variations in some flows, however the measure of benefit is relatively small (*i.e.,* 4.43). Therefore GeoProtect app does not accept the optimizer result, thus no reconfiguration is triggered. In this case, the SDN cost (cost of rules replacement by the controller) dominates. In this scenario, the network state has not much changed which might be the case in most of realistic situations.

TABLE VI
PROBABILITY OF SOURCING AN ATTACK.

Country	Probability	Country	Probability
CN	0.7	PL	0.4
BR	0.6	FR	0.3
SG	0.6	IT	0.3
MY	0.6	ES	0.3
TR	0.5	HK	0.3
ID	0.5	NL	0.2
JP	0.4	IE	0.1

3) Scenario 3: DDoS Attack: Lastly, we emulate a DDoS attack sourced from China by increasing the volume of selected flows (*i.e.,* for 200 prefixes) by a factor of 20. Fig. 5f shows that the total traffic rate of China increases to more than 3 Gbps, over the epoch [60, 75] min. After re-optimization, we observe that the rules for these 200 DDoS prefixes are pushed higher towards the source and placed at node 1, as shown in Fig. 5e and Table V, dropping the attack traffic at the first node before entering the exchange fabric and protecting the expensive bandwidth of the IXP fabric. The rest of rules are distributed almost the same way as before. By comparing Tables V and IV, we see that placement of rules for U.S. and Canada is fairly consistent in two scenarios; for India, about 500 rules are placed further up at nodes 1 and 2; and for China, only 200 rules of attack prefixes are placed at node 1 while others are placed further down the path as they carry lower volumes of traffic. In this case, we achieve a very high benefit of 38.75 (with only 12,550 replacements) which triggers a reconfiguration of the Openflow rules in the network switches by the GeoProtect application.

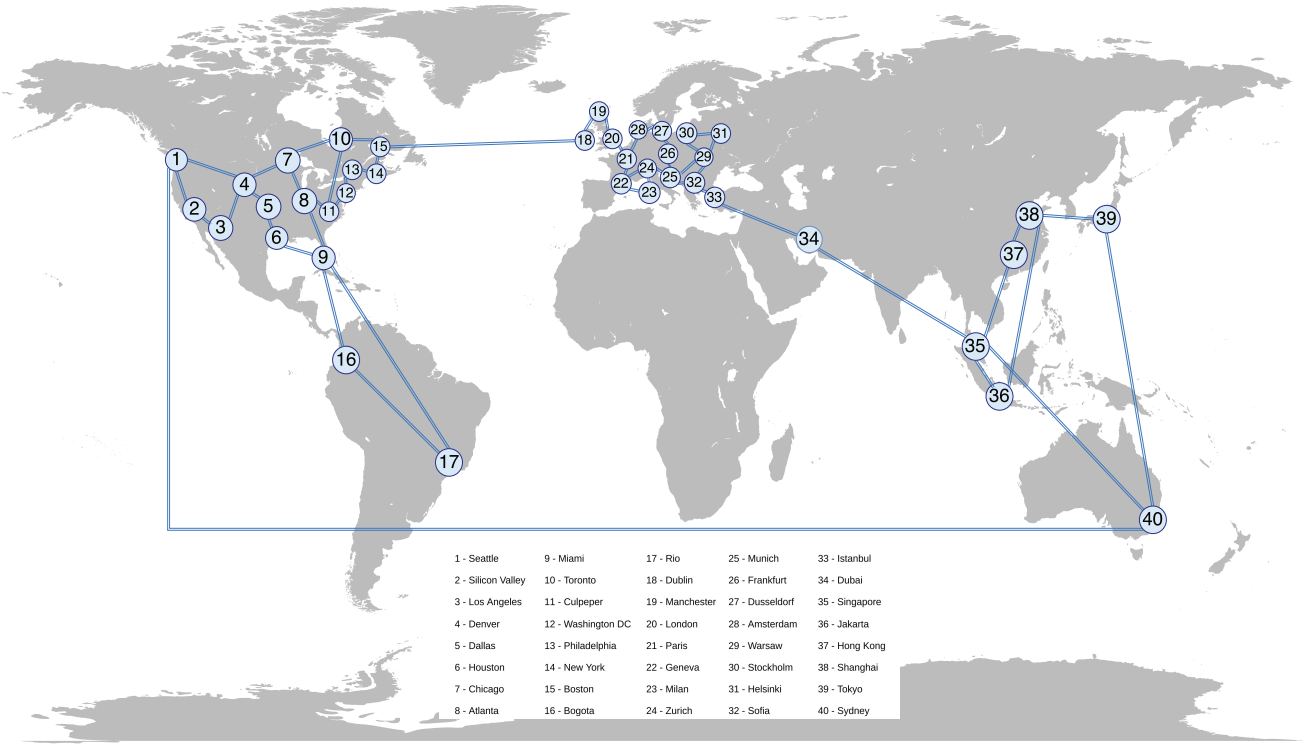


Fig. 6. Large-scale commercial Exchange topology.

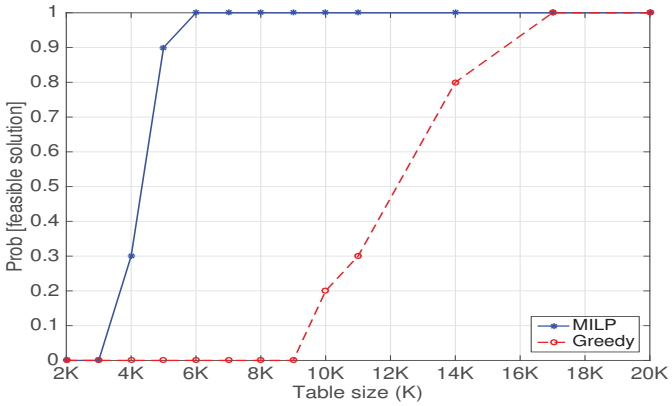


Fig. 7. Probability of finding a feasible solution (MILP vs. Greedy).

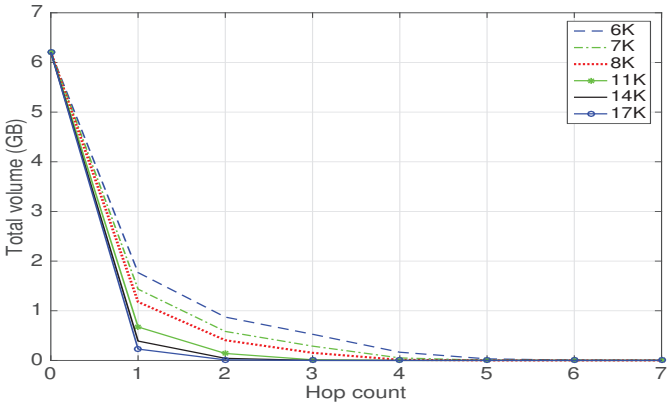


Fig. 8. Volume traversing through nodes by hop length (using MILP).

B. Evaluation in Large Scale IXP Topology

We now evaluate the efficacy of our solution on a large scale IXP topology inspired by Equinix [39], so as to illustrate the performance of our optimization algorithm on a realistic topology. The topology consists of 40 IXP nodes that are spread across the globe connected in a redundant fashion as shown in Fig. 6. We select 7 connected autonomous systems (or enterprises) who express geo-block policies and are connected at nodes 2 (San Francisco, US), 14 (New York, US), 17 (Rio, Brazil), 20 (London, UK), 34 (Dubai, UAE), 35 (Singapore), and 40 (Sydney, Australia). In our experiment, these ASes randomly select sources from a pool of countries to geo-block them. Each source country has a "probability of being chosen" associated with it, carefully selected as per the historical statistics of sourcing Internet attacks [7], [40], [41], and is shown in Table VI. According to security firms [42], half of recent DDoS attacks originated from only ten countries having an insecure infrastructure. The attacks may originate in another country, but are then amplified through other environments. IT infrastructures in these countries tend to have weaker security measures in place, which is why computing resources located therein are used more frequently to commit attacks.

We formulate 7 geo-block policies (for above 7 destination ASes) by selecting a set of sources for each with a given probability. We repeat this process 10 times to select different "policy-sets" (each contains 7 policies), to randomize the selection of sources. When translated into Openflow rules, each policy-set consists of number of rules ranging from 80,000 to 150,000. We vary flow table sizes for each node of our topology starting from a small capacity 2000 where there is no solution from MILP, increasing it by step of 1000 till MILP certainly gives a feasible solution, and changing the step to 3000 till there is no need for MILP due to large

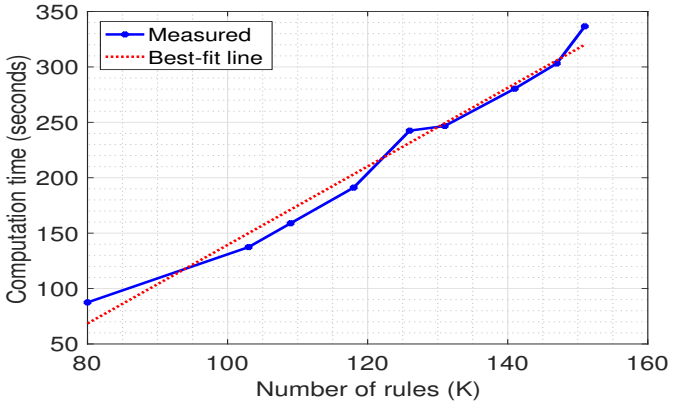


Fig. 9. Time complexity of our optimization as a function of rules count.

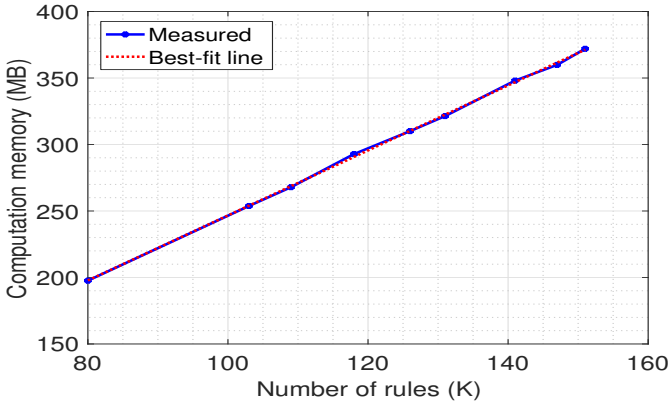


Fig. 10. Memory complexity of our optimization as a function of rules count.

capacity 20000. We next evaluate the efficacy of our optimal MILP algorithm compared to greedy algorithm in finding a feasible solution for placement of rules.

We run both algorithms on each policy set (10 in total) with different flow table sizes. In Fig. 7 we plot the probability of finding a feasible solution for a given table size. The MILP algorithm (shown by solid blue lines) starts finding a feasible solution at table size 4,000 with a probability of 0.3 (*i.e.,* 3 out of 10 policy-sets), while the greedy algorithm (shown by dashed red lines) is incapable of finding a solution until the capacity of table size increases to 10,000. The MILP is able to find solutions for all policy-sets when table size reaches to 6,000 and beyond, while the greedy requires table sizes of at least 17,000 to solve the problem for all policies. The evident reason for out-performance of an optimal algorithm is the correlation and overlap between the various paths in the network. Such overlap is ignored by the short sighted greedy algorithm, but is taken care of by our optimal MILP which considers all possibilities to place the rules. The huge gap between the performance of MILP and greedy algorithm signifies the need of an optimal solution to optimize the use of fabric bandwidth.

We note that when a feasible solution is found by both greedy and optimal algorithms, the computed cost is equal. This means that whenever the greedy algorithm is able to find a solution, it is equivalent to the solution determined by the MILP optimal algorithm. This is an important observation which means that relaxing the constraints on table sizes makes the greedy algorithm to perform the same way as the MILP in finding a feasible solution with equal cost.

Now, we show that how our optimal solution reduces the carriage of unwanted traffic in the fabric and saves bandwidth resources. Unwanted traffic is dropped at a node along their respective path by a rule placed at a node (computed by the optimizer). Obviously, dropping packets of an unwanted flow earlier on its path would save link resources. In other words, high volume traffic should be ideally dropped at lower hop-lengths. Fig. 8 shows the amount of volume carried by all unwanted flows (specified by combination of all policy-sets) along their paths with increasing hop-length. Each curve represents a given table size. Hop-length of zero means traffic is dropped at the first node in the fabric on its path. We observe that higher volume flows are dropped at low hop counts and the amount of unwanted traffic being carried reduces significantly with increasing hop counts. Also, with the increasing table size capacity, more rules can be placed at earlier nodes on the path, reducing the carriage of traffic. It is seen in Fig. 8 that for the table size of 17,000 (shown by circle markers and solid blue lines), no unwanted traffic is carried beyond the hop-length of 2. Increasing the table size capacities to infinity would result in accommodating all rules at the first node on the respective path – achieving the ideal rules placement at zero cost (*i.e.,* dropping all unwanted traffic at the first node).

Scalability of Our Algorithm: In our optimization problem, the number of variables in the objective function is $N.R$ – we have a total of R rules, and each rule is placed at one of the N possible nodes in the topology. The size of inequality and equality constrains is $N^2.R$ and $2N.R^2$ respectively (*i.e.,* it grows quadratically by rules count). We note that the size of constraints is significantly large, and can lead to potential scalability issues. However, the matrices in our constraints are sparse as they contain a large number of zero-valued elements. Therefore, we have reduced the complexity of our algorithm by employing sparse matrices throughout our optimization so that we can both save a significant amount of memory and speed up the processing of our algorithm. In Fig. 9 we plot the computation time of our optimization as a function of rules count. The solid blue line shows the time measured on a machine (running the algorithm) with 4 cores of CPU and 16GB of RAM, and the dashed red line is the best-fit line for our measured data. Note that this plot corresponds to a scenario where the table size is 6K – our measurements for other table sizes match this curve with negligible variations, and are thus not shown. We can see that the time complexity grows linearly in number of rules (*i.e.,* proportional to R for a fixed $N = 40$ in our simulation) – the actual time in seconds depends on available resources on the machine which runs the algorithm. It is also seen that the algorithm takes only 336.6 seconds (*i.e.,* less than 6 minutes) for placement of 151,000 rules. Note that time to mitigate an attack (reaction-time) depends on the epoch interval (15 min.) and the optimization time (6 min.) – depending upon when the attack starts, reaction-time can vary between the optimization time and the optimization time plus the epoch interval (*i.e.,* 6 to 21 minutes in this case). Similarly, Fig. 10 shows that the memory usage of our algorithm to find the optimal rule placement grows

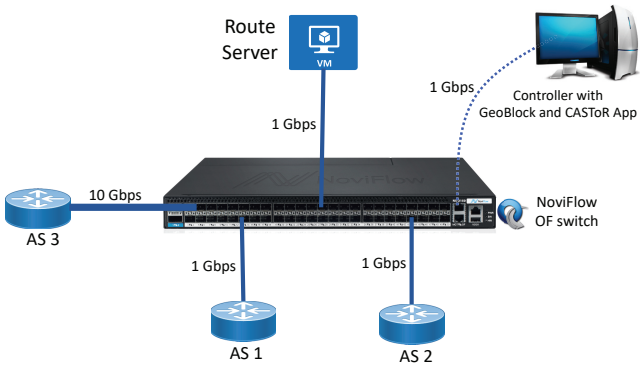


Fig. 11. Experimental setup.

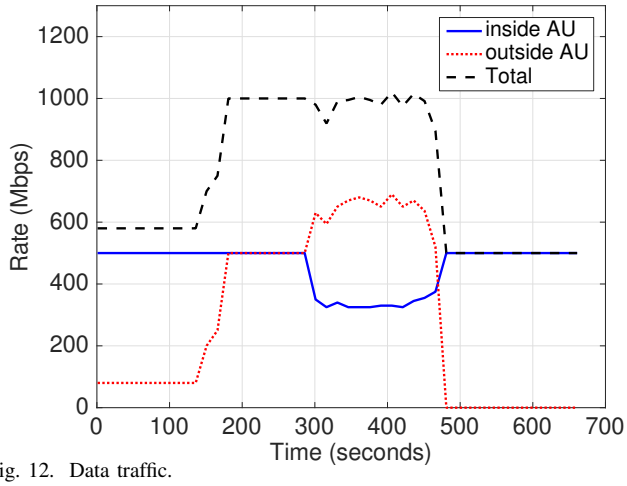


Fig. 12. Data traffic.

linearly in rules count, and we need about 371 MB of memory for handling 151,000 rules in our simulation.

C. Prototype Implementation and Validation

We have implemented our entire solution on the open-source ONOS-based SDN interconnect platform. In this section we explain our prototype components and demonstrate its ability to block attacks in a laboratory testbed.

Geo-Block Application: Our GeoProtect application consists of two main components:

- *Policy Parser*: We have developed REST APIs for our application to receive abstract policies in the form of our defined grammar (discussed in §III). We have used the ANTLR library [38] for defining and parsing our grammar. Our policy parser, written in Java, extracts parsed tokens, breaks down the abstract policies (into blocking, connectivity, and exception policies), and formulates lower-level components of the policy (as described in Fig. 3). This information is then used by the compiler as stated next.
- *Compiler*: Our compiler, written in Java, takes all parsed policies (including exceptions) and formulates the ONOS Intents using the ONOS Intent framework API (`MultiPointToSinglePointIntent` and `FlowObjectiveIntent`). To do this, it uses the topology information provided by ONOS controller and BGP announcements from BGPmon. These intents are then converted into Openflow rules by the ONOS intent framework. Rules are finally installed into the switch by the ONOS controller (as detailed in §III).

Geo-Block Database: Our GeoProtect app maintains an essential database that holds thousands of entries used for mapping the Geo country codes, domain names, and AS numbers associated with IP prefixes. We have implemented this database in SQLite and used the Django back-end framework [43] to support the necessary REST APIs for querying the database. We obtained the geographical IP prefix ownership data from Maxmind [34] and Hurricane Electric BGP toolkit [35]. This data is synced on a weekly, or monthly basis. Our Django back-end implementation also provides configurable APIs that can be used to add extra data (*e.g.,* specific to customers) to the database manually by the administrator. For example, exceptions such as for DNS servers can be added manually and updated on a regular basis.

Prototype Validation: We test our prototype in a laboratory testbed as shown in Fig. 11. The testbed emulates an SDN-enabled exchange point that consists of an OpenFlow switch (*i.e.,* NoviFlow NS1132), an ONOS controller with our GeoProtect app, a route server, and three connected routers (peers) to represent autonomous systems interconnected at the exchange point. Our controller and route server run on two virtual machines in our laboratory cloud environment, and we use Cisco router 3800 series for the border routers of the three peers (*i.e.,* ASes 1, 2, and 3). For traffic generation, we use the Spirent TestCenter, which is a high-precision commercial-grade hardware with a 12-port GE Hypermetrics test module and firmware version 4.24.1026. The capacity of each link between the border routers and the NoviFlow switch is shown in Fig. 11.

To emulate the Australian Census attack, we assume that the Census website and associated infrastructure are within AS1, and attack traffic is sourced from outside Australia via AS3 with very large volume flows representing a form of DDoS. The legitimate traffic (originated from within Australia) is sourced from AS2.

Fig. 12 shows the results of our Census experiment. All traffic originated from AS2 and AS3, and was destined to AS1 where the Census website is located. With the Spirent, we used a combination of public source IP addresses to emulate hosts from inside Australia and hosts outside Australia. At the beginning of our experiment, the traffic rate for inside Australia (shown by solid blue lines) starts at 500 Mbps, and for outside Australia (shown by dotted red lines) it starts at 80 Mbps. In this case, the link connecting the AS1 to the fabric is not oversubscribed, thus no packet loss is observed. At some time between 100s and 200s, we initiated a DDoS attack from outside Australia by increasing the traffic rate sourced from AS3 to 500 Mbps, thus utilizing 100% of the AS1 link. There was still no loss on the legitimate traffic, so we increased the DDoS rate to 1 Gbps at time about 300s. It is clearly seen that the AS1 link became congested and consequently the legitimate traffic streams experienced 30-35% of loss on average.

We then invoked the Geo-Block policy (on behalf of AS1) by which the controller sent flow table updates to the switch dropping all packets with source addresses from outside Aus-

tralia. This resulted in 6,384 rules which were then installed in the switch pipeline that allowed only packets with source addresses corresponding to Australian IP prefixes and all other packets were dropped. The result of this policy is seen in Fig. 12 just before the 500s time marker. The DDoS traffic from outside Australia reduced to zero, the legitimate traffic from within Australia recovered back to 500 Mbps with no loss and the Census service became fully available again. Our system was responsive and scaled to thousands of flows that were pushed to the switch.

VI. Conclusion

Protecting enterprises against DDoS and IP-address spoofing attacks can not be implemented easily in today's networks. This paper proposes a viable security management solution implemented at a software-defined Exchange Point. We have developed an abstract grammar that allows enterprise operators to express their security policies along with a compiler for automatically synthesizing flow rules inserted into the interconnect fabric. We then employed an optimal algorithm to determine the placement of rules across the switch fabric by minimizing the carriage cost of unwanted traffic given the limited table size at each switch. Finally, we evaluated the performance of our scheme via simulation, and demonstrated its efficacy via prototype implementation on the open-source ONOS controller in an SDN testbed. We have deployed our CaSToR platform in two production exchange networks namely Amlight in Florida, and CENIC in California. We also plan to test our geo-protect app in current deployments and evaluate its benefits of security and filtering features.

References

[1] H. Kumar and A. Mercian and S. Banerjee and C. Russell and V. Sivaraman, "Implementing Geo-Blocking and Spoofing Protection in Multi-Domain Software Defined Interconnects," in *Proc. EuroSys XDOMO workshop*, Belgrade, Serbia, Apr 2017.

[2] Arbor Networks. (2017) No end in sight for DDoS attack size growth. https://pages.arbornetworks.com/rs/082-KNA-087/images/WISR_Infographic_NoEndInSight_FINAL.pdf.

[3] "Australia census : Abs website crashes." https://goo.gl/nQxgLb.

[4] "Packet viper : Advanced ip filtering solutions." http://www.packetviper.com.

[5] "Open network operating system," http://onosproject.org/.

[6] "NETSCOUT Arbor's 13th Annual Worldwide Infrastructure Security Report," Arbor Networks, Tech. Rep., 2018.

[7] A. Wang, A. Mohaisen, W. Chang, and S. Chen, "Delving into Internet DDoS Attacks by Botnets: Characterization and Analysis," in *Proc. IEEE/IFIP Dependable Systems and Networks*, Rio de Janeiro, Brazil, June 2015.

[8] Paloalto Networks. (2019) Next-Generation Firewall. [Online]. Available: https://www.paloaltonetworks.com/products/secure-the-network/next-generation-firewall

[9] Fortinet. (2019) FortiGate: Next Generation Firewall. [Online]. Available: https://www.fortinet.com/products/next-generation-firewall.html

[10] IETF. (2018) Next-Generation Firewall Performance Benchmarking Methodology Draft. [Online]. Available: https://datatracker.ietf.org/meeting/101/materials/slides-101-opsec-draft-balarajah-bmwg-ngfw-performance-00

[11] "Next generation geo-ip filtering can be fine tuned to vastly reduce unwanted and malicious traffic," https://goo.gl/VfxEdj.

[12] "Network ingress filtering," https://www.ietf.org/rfc/rfc2827.txt.

[13] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. J. Clark, and E. Katz-Bassett, "SDX: A Software Defined Internet Exchange," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, Aug 2014.

[14] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An Industrial-Scale Software Defined Internet Exchange Point," in *Proc. USENIX NSDI*, Santa Clara, CA, USA, Mar 2016.

[15] J. Stringer, D. Pemberton, Q. Fu, C. Lorier, R. Nelson, J. Bailey, C. N. A. Correa, and C. E. Rothenberg, "Cardigan: Sdn distributed routing fabric going live at an internet exchange," in *Proc. IEEE Symposium on Computers and Communications (ISCC)*, Madeira, Portugal, June 2014.

[16] R. Lapeyrade, M. Bruyère, and P. Owezarski, "OpenFlow-based Migration and Management of the TouIX IXP," in *Proc. IEEE NOMS*, Istanbul, Turkey, April 2016.

[17] K.-K. Yap et al, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, August 2017.

[18] H. Kumar. (2017) castorlive. [Online]. Available: https://github.com/Vijay-Sivaraman-Research-Group/castorlive/tree/onos-1.8

[19] M. Chiesa, C. Dietzel, G. Antichi, M. Bruyere, I. Castro, M. Gusat, T. King, A. W. Moore, T. D. Nguyen, P. Owezarski, S. Uhlig, and M. Canini, "Inter-Domain Networking Innovation on Steroids: Empowering IXPs with SDN Capabilities," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 102–108, October 2016.

[20] M. Chiesa, D. Demmler, M. Canini, M. Schapira, and T. Schneider, "Towards Securing Internet eXchange Points Against Curious onlooKers," in *Proc. Applied Networking Research Workshop*, July 2016.

[21] C. Dietzel, A. Feldmann, and T. King, *Blackholing at IXPs: On the Effectiveness of DDoS Mitigation in the Wild*, March–April 2016.

[22] "Project boulder nbi (north bound interface)," http://opensourcesdn.org/projects/project-boulder-intent-northbound-interface-nbi.

[23] "Onos intent framework," https://wiki.onosproject.org/display/ONOS/Intent+Framework.

[24] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using Graphs to Express and Automatically Reconcile Network Policies," in *Proc. ACM SIGCOMM*, London, United Kingdom, August 2015.

[25] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software defined networks," in *Proc. USENIX NSDI*, Lombard, IL, USA, April 2013.

[26] "Pyretic framework," http://frenetic-lang.org/pyretic/.

[27] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proc. ACM SIGCOMM*, Florianopolis, Brazil, August 2016.

[28] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing rules placement in openflow networks: trading routing for better efficiency," in *HotSDN*, Chicago, US, Aug 2014.

[29] S. Zhang, F. Ivancic, C. Lumezanu, A. Gupta, and S. Malik, "An adaptable rule placement for software-defined networks," in *Dependable Systems and Networks (DSN)*, Atlanta, US, Sep 2014.

[30] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. INFOCOM*, Apr 2013.

[31] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'One Big Switch' Abstraction in Software-Defined Networks," in *Proc. ACM CoNEXT*, Dec 2013.

[32] Himal Kumar et al, "A software defined flexible inter-domain interconnect using onos," in *Proc. EWSDN*, The Hague, Netherlands, Oct 2016.

[33] "17th annual global lambdagrid workshop," https://www.glif.is/meetings/2017/tech/mambretti-starlight.pdf.

[34] "Maxmind : Geoip databases," https://www.maxmind.com/en/home.

[35] "Hurricane electric bgp toolkit," http://bgp.he.net/.

[36] "Bgpmon : Tool for monitoring bgp updates," http://www.bgpmon.io/.

[37] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, "Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing," in *Proc. NDSS*, San Diego, California, USA, 2003.

[38] "Antlr language parsing tool," http://www.antlr.org/.

[39] "Equinix internet exchange global locations," http://www.equinix.com.au/locations/.

[40] "Top 10 ddos attack source countries," https://mybroadband.co.za/news/security/188774-top-10-ddos-attack-source-countries.html.

[41] "Share of global denial of service attack traffic during 4th quarter 2016, by originating country," https://www.statista.com/statistics/440582/ddos-attack-traffic-by-originating-country/.

[42] "White paper: The Top 10 DDoS Attack Trends," Imperva, Tech. Rep., 2015.

[43] "Django web framework," https://www.djangoproject.com/.

Himal Kumar received his Master of Philosophy degree from the University of New South Wales (UNSW) in 2017 and Bachelors from the Indian Institute of Technology (IIT Patna) in 2014. His expertise has been in applying SDN/NFV technology in the areas of Internet Exchange Points (IXPs), High-Speed Flow Classification and NFV for middle-boxes. Himal has years of experience in building production grade SDN solutions and has engineered multiple SDN/NFV applications deployed and trialled in research and production networks. He is also a part of open-source SDN community and has contributed to applications in ONOS and ODL.

Hassan Habibi Gharakheili received his B.Sc. and M.Sc. degrees of Electrical Engineering from the Sharif University of Technology in Tehran, Iran in 2001 and 2004 respectively, and his Ph.D. in Electrical Engineering and Telecommunications from the University of New South Wales in Sydney, Australia in 2015. He is currently a lecturer at the University of New South Wales in Sydney, Australia. His current research interests include programmable networks, learning-based networked systems, and data analytics in computer systems.

Craig Russell received his Ph.D. in Applied Mathematics from Macquarie University, Sydney in 1997. He is currently a Principal Research Engineer at CSIRO Data61 and has previously held commercial roles in the telecommunications and software industries. He has design, implementation and operational experience in a wide range of advanced telecommunications equipment and protocols as well as experience in developing software applications. His research interests are in software-defined networking and the application of machine learning techniques to solve problems in network security.

Vijay Sivaraman received his B. Tech. from the Indian Institute of Technology in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000. He has worked at Bell-Labs as a student Fellow, in a silicon valley start-up manufacturing optical switch-routers, and as a Senior Research Engineer at the CSIRO in Australia. He is now a Professor at the University of New South Wales in Sydney, Australia. His research interests include Software Defined Networking, network architectures, and cyber-security particularly for IoT networks.