Introducing SmartNICs in Server-Based Data Plane Processing: The DDoS Mitigation Use Case

(Article begins on next page)

Introducing SmartNICs in Server-based Data Plane Processing: the DDoS Mitigation Use Case

Sebastiano Miano*, Roberto Doriguzzi-Corin†, Fulvio Risso*, Domenico Siracusa†, Raffaele Sommese‡
*Politecnico di Torino, Department of Computer and Control Engineering, Torino, Italy
†CREATE-NET, Fondazione Bruno Kessler, Trento, Italy
‡University of Twente, Design and Analysis of Communication Systems, Enschede, The Netherlands

***Abstract*—In the recent years, the complexity of the network data plane and their requirements in terms of agility has increased significantly, with many network functions now implemented in software and executed directly in datacenter servers. To avoid bottlenecks and to keep up with the ever increasing network speeds, recent approaches propose to move the software packet processing in kernel space using technologies such as eBPF/XDP, or to offload (part of it) in specialized hardware, the so called SmartNICs. This paper aims at guiding the reader through the intricacies of the above mentioned technologies, leveraging SmartNICs to build a more efficient processing pipeline and providing concrete insights on their usage for a specific use case, namely, the mitigation of Distributed Denial of Service (DDoS) attacks. In particular, we enhance the mitigation capabilities of edge servers by transparently offloading a portion of DDoS mitigation rules in the SmartNIC, thus achieving a balanced combination of the XDP flexibility in operating traffic sampling and aggregation in the kernel, with the performance of hardware-based filtering.**

We evaluate the performance in different combinations of host and SmartNIC-based mitigation, showing that offloading part of the DDoS network function in the SmartNIC can indeed optimize the packet processing but only if combined with additional processing on the host kernel space.

I. Introduction

With the recent trend of "network softwarization", promoted by emerging technologies such as Network Function Virtualization (NFV) and Software Defined Networking (SDN), system administrators of data center and enterprise networks have started to replace dedicated hardware-based middleboxes with virtualized Network Functions (NFs) running on commodity servers and end hosts [1]–[6]. This radical change has facilitated the provisioning of advanced and flexible network services, ultimately helping the system administrators to cope with the rapid changes on service requirements and networking workloads.

Unfortunately, the ever growing network capacity installed in data center and enterprise networks requires a highly flexible low-latency network processing, which is hardly achievable with standard packet processing mechanisms implemented in the operating systems of servers and end-hosts. Common solutions rely on kernel bypass approaches, such as DPDK [7] and Netmap [8], which map the network hardware buffers directly to user space memory, hence bypassing the operating system. Although these technologies bring an unquestionable performance improvement, they also have two major limitations. First, they take the ownership of one (or more) CPU cores, thus permanently stealing precious CPU cycles to other tasks (NFs deployed on the servers, or user applications running on the end hosts). Second, they require to install additional kernel modules or to update the network card driver, operations that are not always possible in production networks.

Recent technologies such as eBPF [9], [10] and eXpress Data Path (XDP) [11] offer excellent processing capabilities without requiring to permanently allocate dedicated resources in the host; eBPF programs combined with XDP are executed at the earliest level of the Linux networking stack, directly upon the receipt of a packet and immediately after the driver RX queues. Furthermore, eBPF/XDP are included in vanilla Linux kernels, hence avoiding the need to install custom kernel modules or additional device drivers.

To further reduce the workload on the precious general-purpose CPU cores of the servers, system administrators have resumed the old idea of introducing *programmable* intelligent networking adapters (a.k.a., *SmartNICs*) in their servers [12], [13], hence combining the flexibility of software network functions with the improved performance of the hardware NIC acceleration. SmartNICs offer hardware accelerators that enable to partially (or fully) offload packet processing functions; examples include load balancing [14], key-value stores [15] or more generic flow-level network functions [16], [17]. On the other hand, SmartNICs may present additional challenges due to their limited memory and computation capabilities compared to current high-performance servers.

In this paper we consider the potential of exploiting SmartNICs on a specific use case, i.e., to mitigate volumetric DDoS attacks, which are considered as one of the major threats in today's Internet, accounting for the 75.7% of the total DDoS attacks [18]–[20]. While the *detection* of DDoS attacks is a largely studied problem in the literature with several algorithms proposed to rapidly and efficiently detect an ongoing attack, in this paper we focus on the challenges related to the DDoS attack *mitigation*; in particular, we explore how the recent advances on the host data-plane acceleration can be used to adequately handle the large speeds required by today's networks.

This paper provides the following contributions. First, we analyze the various approaches that can be used to design an efficient and cost-effective DDoS mitigation solution. As generally expected, our results show that offloading the mitigation task to the programmable NIC yields significant performance improvements; however, we demonstrate also that, due to the memory and compute limitations of current SmartNIC technologies, a fully offloaded solution may lead to deleterious performance. Second, as a consequence of the previous findings, we propose the design and implementation of a hybrid mitigation pipeline architecture that leverages the flexibility of eBPF/XDP to handle different type of traffic and attackers and the efficiency of the hardware-based filtering in the SmartNIC to discard traffic from malicious sources. Third, we present a mechanism to transparently offload part of the DDoS mitigation rules into the SmartNIC, which takes into account the most aggressive sources, i.e., the ones that largely impact on the mitigation effectiveness.

This rest of the paper is structured as follows. Section II presents a high-level overview of eBPF and XDP, together with the SmartNIC and TC Flower, the flow classifier of the Linux traffic control kernel subsystem. Section III analyzes the different approaches that can be used to build an efficient DDoS mitigation solution. Section IV presents the design of an architecture that uses the above mentioned technologies to both detect and mitigate DDoS attacks, including the offloading algorithm adopted to install the rules into the SmartNIC (Section IV-A1), while keeping the flexibility and improved performance of the in-kernel XDP packet processing. Finally, Section V provides the necessary evidence to the previous findings, Section VI briefly discusses the related works and Section VII concludes the paper.

II. Background

A. extended Berkeley Packet Filter (eBPF)

The extended Berkeley Packet Filter (eBPF) is an enhanced version of the original BPF virtual machine [21], originally developed as kernel packet filtering mechanism for the BSD operating system and used by tools such as `tcpdump`. Compared to the original version, eBPF enables the execution of custom bytecode (either interpreted or compiled just-in-time) at various points of the Linux kernel in a safe manner. Furthermore, thanks to the support from the Clang/LLVM compiler, eBPF programs can be written in a restricted-C language, which is then compiled into the corresponding eBPF object file that can be loaded into the kernel through the apposite `bpf()` system call. In addition to the improved and enriched instruction set, eBPF offers several pre-defined data structures (e.g., hash map, lru map, array) that can be read/written from either kernel or userspace program, hence providing the possibility to modify the behavior of an eBPF program based upon dynamically changing operating conditions. Moreover, it provides helper functions that can either be used to implement complex features that may not be feasible in the eBPF restricted-C, or to interact with kernel-level functionalities. Finally, eBPF programs can be cascaded in order to create larger service chains. The above additional capabilities allow eBPF to provide its functions in a broad range of kernel-level use cases, such as tracing, security and networking. In particular, in the latter case, this special-purpose event-driven virtual machine enables arbitrary packet processing on incoming/outgoing traffic directly in the Linux kernel, with the possibility to re-configure the existing eBPF programs to adapt to the (dynamically changing) operating conditions. This provides an unique option for flexibility and efficiency that was not available before.

1) eXpress Data Path (XDP): Networking eBPF programs can be attached to different points of the Linux stack. Starting from Linux kernel v4.8, the eXpress Data Path (XDP) provides the possibility to execute those programs at the lowest level of the TCP/IP stack, in the NIC driver itself, before the allocation of costly kernel data structures (e.g., `sk_buff`), thus achieving the best possible packet processing performance in the kernel stack. As consequence, they represent the best choice to detect and drop malicious packets with minimal consumption of the host CPU resources, and will represent one of the key technologies exploited in this paper.

B. SmartNICs

Smart Network Interface Cards (SmartNICs) are intelligent adapters used to boost the performance of servers by offloading (part of) the network processing workload from the host CPU to the NIC itself [22]. Although the term SmartNIC is being widely used in the industry and academic world, there is still some confusion over the precise definition. We consider traditional NICs the devices that provide several pre-defined offloaded functions (e.g., transmit/receive segmentation offload, checksum offload) without including a fully programmable processing path, e.g., which may involve the presence of a general-purpose CPU on board. In our context, a *SmartNIC* is a NIC equipped with a fully-programmable system-on-chip (SoC) multi-core processor that is capable to run a fully-fledged operating system, offering more flexibility and hence potentially taking care of any arbitrary network processing task. This type of SmartNIC can also be enhanced with a set of specialized hardware functionalities that can be used to accelerate specific class of functions (e.g., OpenvSwitch data-plane) or to perform generic packet and flow-filtering. On the other hand, they have limited compute and memory capabilities, making not always possible (or efficient) to completely offload all types of tasks. Furthermore, SmartNICs feature their own operating system and therefore may have to be handled separately from the host. For instance, offloading a network task to the SmartNIC may require the host to have multiple interactions with the card, such as to compile and inject the new eBPF code, to execute additional commands (either on the host, or directly on the card) to exploit the available features such as configure hardware co-processors. Finally, no current standard exist to interact with SmartNICs, hence different (and often proprierary) methods have to be implemented when the support of several manufacturers is required.

C. TC Flower

The Flow Classifier is a feature of the Linux Traffic Control (TC) kernel subsystem that provides the possibility to match, modify and apply different actions to a packet based on the flow it belongs to. It offers a common interface for hardware vendors to implement an offloading logic within their devices; when a TC Flower rule is added, active NIC drivers check if that rule is supported in hardware; in that case the rule is pushed to the physical card, causing packets to be directly matched in the hardware device, hence resulting in greater throughput and a decrease of the host CPU usage.

TC Flower represents a promising technology that can hide the differences between different hardware manufacturers, but it not able (yet) to support all the high-level features that may be available in modern SmartNICs.

III. DDoS Mitigation: approaches

Once a DDoS attack is detected, efficient packet dropping is a fundamental part of a DDoS attack mitigation solution. In a typical DDoS mitigation pipeline, a set of mitigation rules are deployed in the server's data plane to filter the malicious traffic. The strategy used to block the malicious sources may be determined by several factors such as the characteristics of the server (e.g., availability of a SmartNIC, its hardware capabilities), the characteristics of the malicious traffic (e.g., number of attackers) or the type and complexity of the rules that are used to classify the illegitimate traffic. In particular, we envision the following three approaches.

1) Host-based mitigation: In this case all traffic (either malicious or legitimate) is processed by the host CPU, which drops incoming packets that match a given blacklist of malicious sources; this represents the only viable option if the system lacks of any underlying hardware speedup.

All the host-based mitigation techniques and tools used today fall in two different macro-categories depending on whether packets are processed at kernel or user-space level.

Focusing on Linux-based system, the first category includes `iptables` and its derivatives, such as `nftables`, which represent the main tools used to mitigate DDoS attacks. It allows to express complex policies to the traffic, filtering packets inside the `netfilter` subsystem. However, the deep level in the networking stack where the packet processing occurs causes poor performance when coping with increasing speed of the today's DDoS attacks, making this solution practically unfeasible, as demonstrated in Section V.

As opposite to kernel-level processing, a multitude of fast packet I/O frameworks relying on specialized NIC/networking drivers and user-space processing have been built over the past years. Examples such as Netmap [8], DPDK [7], PF_RING ZC [23] rely on a small kernel component that maps the NIC device memory directly to user space, hence making it directly available to (network-specialized) userland applications instead of relying on normal kernel data-path processing. This approach provides huge performance benefits compared to the standard kernel packet processing but incurs in several non-negligible drawbacks. First of all, these frameworks require to take the exclusive ownership of the NIC, so that all packets received are processed by the userspace application. This means that, in a DDoS mitigation scenario, packets belonging to legitimate sources have to be inserted back into the kernel, causing unnecessary packet copies that slow down the performance[1]. Furthermore, these frameworks require the fixed allocation of one (or more) CPU cores to the above programs, independently from the presence of an ongoing attack, hence reducing the performance-cost ratio, as precious CPU resources are no longer available for normal processing tasks (e.g., virtual machines).

XDP can be considered as a mix of the previous approaches. It is technically a kernel-space framework, although XDP programs can be injected from userspace to the kernel, after guaranteeing that all security properties are satisfied. XDP programs are executed in the kernel context but as early as possible, well before the `netfilter` framework, hence providing an improvement of an order of magnitude compared to `iptables`. The adoption of XDP to implement packet filtering functionalities has grown over the years; *(i)* its perfect integration with the Linux kernel makes it more efficient to pass legitimate packets up to the stack, *(ii)* its simple programming model makes it easy to express customized filtering rules without taking care of low-level details such as required by common user-space framework and *(iii)* its event-driven execution gives the possibility to consume resources only when necessary, providing a perfect trade-off between performance and CPU consumption.

2) SmartNIC-based mitigation: If the server is equipped with a SmartNIC, an alternative approach would be to offload the entire mitigation task to this device. This enables to dedicate all the available resources on the host CPU to the target workloads, operating only on the legitimate traffic, freeing the host CPU from spending precious CPU cycles in the mitigation.

However, although SmartNICs (by definition) support arbitrary data path processing, they often differ on how this can be achieved. Possible options range from running a custom executable, which should already be present on the card, to dynamically inject a new program created on the fly, e.g., thanks to technologies such as XDP or P4, or to directly compile those programs into the hardware device [24]. This makes more cumbersome the implementation of offloading features that run on cards from multiple manufacturers.

In our context, we envision two different options: (*i*) exploit any hardware filter (if available) in the SmartNIC and, if the number of blacklisted addresses exceeds the capability of the hardware (which may be likely, given the typical size of the above structure), block the rest of the traffic with a custom dropping program (e.g., XDP) running on the NIC CPU; (*ii*) block all the packets in software, running entirely on the SmartNIC CPU, e.g., in case the card does not have any hardware filtering capability. In both cases, the surviving (benign) traffic is redirected to the host where the rest of server applications are running. An evaluation of the above possibilities will be carried out in Section V.

3) Hybrid (SmartNIC + XDP Host): An alternative strategy that combines the advantages of the previous approaches would be to adopt a hybrid solution where part of the malicious traffic is dropped by the SmartNIC (reducing the overhead on the host's CPU) and the remaining part is handled on the host, possibly leveraging the much greater processing power available in modern server CPUs compared to the one available in embedded devices.

[1] It is worth mentioning that Netmap has a better kernel integration compared to DPDK; in fact, it is possible to inject packets back into the kernel by just passing a pointer, without any copy. However, it is still subjected to a high CPU consumption compared to eBPF/XDP.

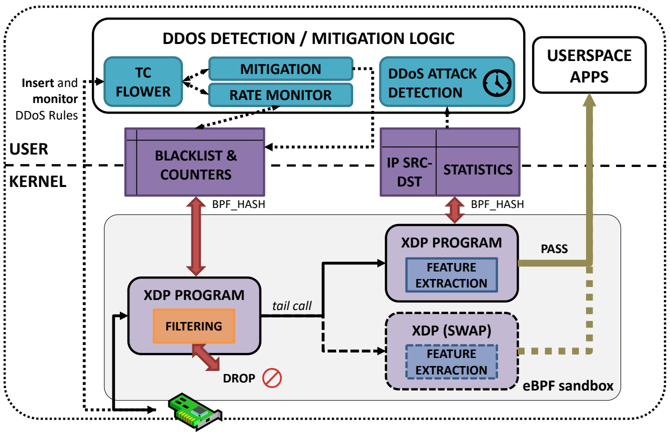


Fig. 1: High-level architecture of the system.

In this scenario, we exploit the fixed hardware functions commonly available in the current SmartNICs to perform stateless matching on selected packet fields and apply simple actions such as modify, drop or allow packets. To avoid redirecting all the traffic to the (less powerful) SmartNIC CPU, we could let it pass through the above hardware tables (where the match/drop is performed at line rate) and forward the rest of the packets to the host, where the remaining part of the mitigation pipeline is running. However, given the limited number of entries often available in the above hardware tables, which are not enough to contain the large number of mitigation rules needed during a large DDoS attack, the whole list of dropping targets is partitioned between the NIC and the host dropping program (e.g., XDP). This requires specific algorithms to perform this splitting, which should keep into account the difference in terms of supported rules and their importance. Interesting, this scenario in which the companion filtering XDP program is executed in the server is also compatible with some traditional NICs that support fixed hardware traffic filtering, such as Intel cards with Flow Director[2]. In this case, the mitigation module can use the card-specific syntax (e.g., Flow Director commands) to configure filtering rules, with the consequent decrease of the filtering processing load in the host.

IV. Architecture and Implementation

This section presents a possible architecture that can be used to compare the previous three approaches in the important use case of the DDoS mitigation, enabling a fair comparison of their respective strength and weaknesses in the implementation of an efficient and cost-effective mitigation pipeline. In particular, we present the different components constituting the proposed architecture (shown in Figure 1) and their role, together we some implementation details that result from the use of the assessed technologies.

[2]The Flow Director is an Intel feature that supports advanced filters and packet processing in the NIC; for this reason it is often used in scenarios where packets are small and traffic is heavy (e.g., DoS attacks).

A. Mitigation

The first program encountered in the pipeline is the *filtering* module, which matches the incoming traffic against the list of blacklisted entries to drop packets coming from malicious sources; surviving packets are redirected to the host where additional (more advanced) checks can be performed before redirecting packets directly to the next program in the pipeline (i.e., the *feature extraction*).

Although our architecture is flexible enough to instantiate the filtering program in different locations (e.g., SmartNIC, Host, and even partitioned across the two above), at the beginning we instantiate an XDP *filtering* program in the host in order to obtain the necessary traffic information and decide the best mitigation strategy. If the userspace DDoS *mitigation* module recognizes the availability of the hardware offload functionality in the SmartNIC, it starts adding the filtering rules into the hardware tables, causing malicious packet to be immediately dropped in hardware. However, since those tables have often a limited size (typically ~1-2K entries), we place the most active top-K malicious talkers in the SmartNIC hardware tables, where K is the size of those tables, while the remaining ones are filtered by the XDP program running either on the SmartNIC CPU or on the host, depending on a configuration option that enables us to compare the results with different operating conditions.

1) Offloading algorithm: The selection of the top-K malicious talkers that are most appropriate for hardware offloading is carried out by the *rate monitor* module, which computes a set of statistics on the dropped traffic and applies a hysteresis-based function to predict the advantages of possibly modifying the list of offloaded rules that are active in the SmartNIC. In fact, altering this list requires either computational resources or time (in our card a single rule update may require up to 2 *ms*), which may be unnecessary if the rank of the new top-K rules does not effectively impact on the mitigation effectiveness.

The pseudo-code of our algorithm is shown in Listing 1. First, it computes a list of the *global* top-K sources, which contains both SmartNIC and XDP entries sorted in *descending* order according to their rate, and a second list containing only the offloaded entries, i.e., the ones present in the SmartNIC hardware tables, which is arranged in *ascending* order. Next, it computes the difference of the above lists, resulting in two lists containing two disjoint set of elements; the first list contains all the *candidate* rules that are not yet in the SmartNIC and the second list includes the SmartNIC entries that are not in the top-K anymore. At this point, starting from the first element of the former list, it calculates the possible benefit obtained by removing the first entry of the second list (given by the ratio between the rate of the two entries) and inserting this new entry in the SmartNIC; if the value is greater than a certain threshold, the entry is moved into the *offloaded* list and the algorithm continues with the next entry. This *threshold* is adjusted according to the current volume of DDoS traffic and it is inversely proportional to it; this avoids unnecessary changes in the top-K SmartNIC list when the traffic rate is low (compared to the maximum achievable rate), which may bring a negligible improvement. On the other hand, it increases the

Algorithm 1 Offloading algorithm

Input: K, the max # of supported SmartNIC entries
Output: $\upsilon'_k \leftarrow$ The list of SmartNIC entries.
$\gamma_k \leftarrow$ TOP-K Global entries
$\upsilon_k \leftarrow$ TOP-K SmartNIC entries
SORTDESCENDING(γ_k)
SORTASCENDING(υ_k)
$\gamma'_k \leftarrow \gamma_k$ - υ_k ▷ Remove already offloaded entries
$\upsilon'_k \leftarrow \upsilon_k$ - γ_k ▷ List of non TOP-K rules
for each $\gamma'_{i,k} \in \gamma'_k$ **do**
  $\beta_i \leftarrow$ OFFLOADGAIN($\gamma'_{i,k}, \upsilon'_{i,k}$)
  if $\beta_i \geq threshold$ **then**
    $\upsilon'_k \leftarrow \upsilon'_k - \upsilon'_{i,k}$ ▷ Remove old entry from offload list
    $\upsilon'_k \leftarrow \upsilon'_k + \gamma'_{i,k}$ ▷ Add new entry into offload list
  end if
end for

update likelihood when the volume of traffic is close to the maximum achievable rate; in this scenario, where the system is overloaded, mitigating even slightly more aggressive talkers may introduce substantial performance benefits.

B. Feature extraction

Although not strictly belonging to the mitigation pipeline, the *feature extraction* module monitors the incoming traffic and collects relevant parameters required by the mitigation algorithm (e.g., counting the number of packets for each combination of source and destination hosts). Being placed right after the mitigation module, it receives all the (presumed) benign traffic that has not been previously dropped so that can be further analyzed and then passed up to the target applications. XDP represents the perfect technology to implement this component since it provides *(i)* the low overhead given by the kernel-level processing and *(ii)* the possibility to dynamically change the behavior of the system by re-compiling and re-injecting (in the kernel) an updated program when we require the extraction of a different set of features . Moreover, XDP offers the possibility to export the extracted information into specific key-value data structures shared between the kernel and userspace (i.e., where the DDoS attack detection algorithm is running) or to directly send the entire packet up to userspace if a more in-depth analysis is needed.

In the former case, data are stored in a per-CPU eBPF hash map, which is periodically read by the userspace *attack detection* application. Since multiple instances of the same XDP program are executed in parallel on different CPU cores, each one processing a different packet, the use of a per-CPU map guarantees very fast access to data thanks to its per-core dedicated memory; consequently data are never realigned with the other caches present on other CPU cores, avoiding the cost of cache synchronization. As result, each instance of the *feature extraction* works independently, saving the statistics of each IP source/destination on its own private map. In the latter case, a specific eBPF helper is used to copy packets to a `perf` event ring buffer, which is then read by the userspace application.

Analysis and Aggregation. Computed traffic statistics are retrieved from each kernel-level hash-map, aggregated by the companion userspace application and saved in memory for further processing. However, this process was found to be relatively slow; our tests report an average of $30\mu s$ to read a single entry from the eBPF map, requiring more than ten seconds to process the entire dataset in case of large DDoS attacks (e.g., ~300K entries). In fact, eBPF does not provide any possibility to read an entire map within a single `bpf()` system call, hence requiring to read each single value separately. As consequence, to guarantee coherent data to the userspace detection application, we should lock the entire table while reading the values, but this would result in the impossibility for the kernel to process the current incoming traffic for a considerable amount of time.

To avoid the above problem, we adopted a *swappable dual-map* approach, in which the userspace application reads data from a first eBPF map that represents a snapshot of the traffic statistics at a given time, while the XDP program computes the traffic information for the incoming packets received in the the previous timespan, and saved in a second map. This process is repeated every time the periodic user-space *detection* process is triggered, hence allowing the detection algorithm to always work with consistent data. From the implementation point of view, we opted for a swappable *dual-program* approach instead of a swappable *dual-map* because of its reduced swapping latency. We create two *feature extraction* XDP programs, each one with its own hash-map, and swap them atomically by asking the *filtering* module to dynamically update the address of the next program in the pipeline, which basically means updating the target address of an assembly `jump` instruction.

C. Detection

The identification of a DDoS attack is performed by the *detection* module, which operates on the traffic statistics presented in the previous section and exploits the retrieved information to identify the right set of malicious sources, which are then inserted in the blacklist map used by the *filtering* module to drop the traffic.

Since the selection of the best mitigation algorithm is out of the focus of this paper, we provide here only a small description of the possible choices that, however, need to be carefully selected depending on the characteristics of the environment and the type of workloads running on the end-hosts. In fact, different approaches are available [19], [25] falling in two main categories: *(i)* anomaly-based detection mechanisms such as entropy-based approaches [26]–[28], used to detect variations in the distribution of traffic features observed in consecutive timeframes and *(ii)* signature-based approaches that employ a-priori knowledge of attack signatures to match incoming traffic and detect intrusions.

It is important to note that the type of detection algorithm may influence the exported traffic information on the *feature extraction* module; however, thanks to the excellent programmability of XDP we can change the behavior of the program without impacting on the rest of the architecture.

D. Rate Monitor

Sometimes, a given detection algorithm may erroneously detect some legitimate sources as attackers. To counter this situation, a specific mechanism is used to eliminate from the blacklist a source that is no longer considered malicious, e.g., because it was considered an attacker by mistake or because it does no longer participate to the attack. This task is performed by the *rate monitor*, which starts from the *global* list of blacklisted addresses, sorted according to their traffic volume, and examines the entries that are at the bottom of the list (i.e., the ones sending less traffic), comparing them with a threshold value; if the current transmission rate of the source under consideration is below the threshold, defined as the highest rate of packets with the same source observed under normal network activity, it is removed from the blacklist. In case the host is removed by mistake, the detection algorithm will re-add to the list of malicious sources in the next iteration.

V. Performance evaluation

This section provides an insight of the benefits of SmartNICs in the important use case of DDoS mitigation. First, it outlines the test environment and the evaluation metrics; then, exploiting the previously described architecture, it analyzes different approaches that exploit SmartNICs and/or other recent Linux technologies such as eBPF/XDP for DDoS mitigation, comparing with the performance achievable with commonly used Linux tools (i.e., *iptables*).

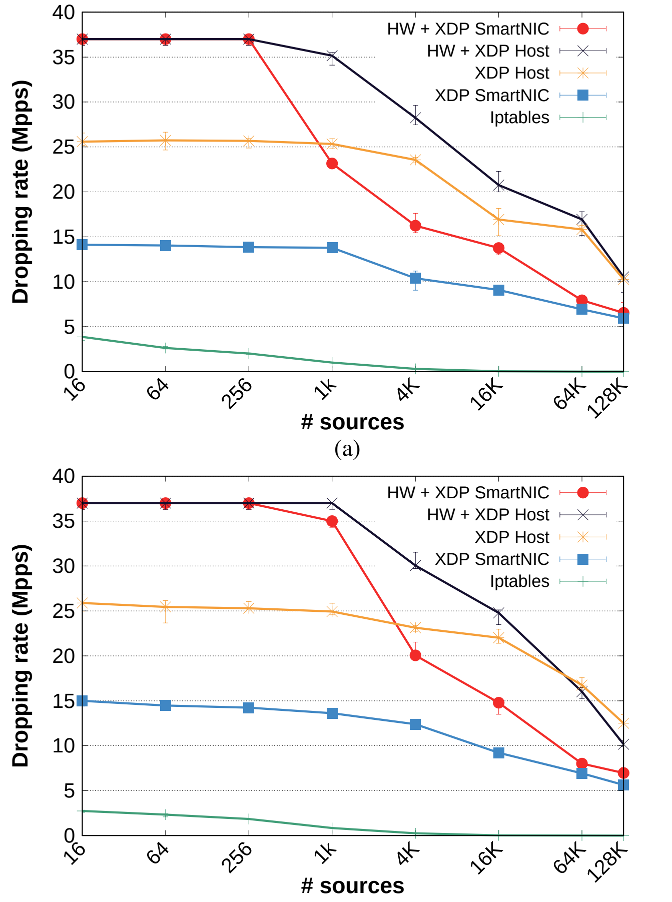
A. Test environment

Our testbed includes a first machine used as packet generator, which creates a massive DDoS attack with an increasing number of attack sources, and a second server running the DDoS mitigation pipeline. Both servers are equipped with an Intel Xeon E3-1245 v5 with a quad-core CPU @3.50GHz, 8MB of L3 cache and two 16GB DDR4-2400 RAM modules, running Ubuntu 18.04.2 LTS and kernel 4.15. The two machines are linked with two 25Gbps SmartNICs, with each port directly connected to the corresponding one of the other server.

We used *Pktgen-DPDK* v3.6.4 and *DPDK* v19.02 to generate the UDP traffic (with small 64B packets) simulating the attack. We report the dropping rate of the system and the CPU usage, which are the two fundamental parameters to keep into account during an attack. We also measure the capability of the server to perform real work (i.e., serve web pages) while under attack, comparing the results of the different mitigation approaches. In this case, the legitimate traffic is generated using the open-source benchmarking tool `weighttp`, which creates a high number of parallel TCP connections towards the device under test; in this case we count only the successfully completed TCP sessions.

B. Mitigation performance

The first test measures the ability of the server to react to massive DDoS attacks that involve an increasing number of sources (i.e., *bots*), showing the performance of different mitigation approaches in terms of dropping rate (Mpps) and CPU consumption. We generate 64B UDP packets at line-rate at 25Gbps (i.e., 37.2Mpps); we consider both a scenario



(a) (b)

Fig. 2: Dropping rate with an increasing number of attackers. *(a)*: uniformly distributed traffic; *(b)*: traffic normally distributed among all sources.

where the traffic is uniformly distributed among all sources (Figure 2a) and a situation where the traffic generated by each source follows a Gaussian distribution (Figure 2b). In addition, we report the CPU consumption for the first test (uniform distribution) in Figure 3.

1) Iptables: One of the most common approaches for DDoS attacks mitigation relies on `iptables`, a Linux tool anchored to the *netfilter* framework that can filter traffic, perform network address translation and manipulate packets. For this test we deployed all the rules containing the source IPs to drop in the `PREROUTING` *netfilter* chain, which provides higher efficiency compared to the more common `INPUT` chain, which is encountered later in the networking stack. Figure 2a and 2b show how the dropping rate of *iptables* are rather limited, around 2.5-4.5Mpps, even with a relatively small number of attack sources, making this solution incapable of dealing with the massive DDoS attacks under consideration. This is mainly given by the linear matching algorithm used by *iptables*, whose performance degrade rapidly when an increasing number of rules are used, leading to a throughput almost equal to zero with more than 4K rules. The CPU consumption (Figure 3) confirms this limitation; using *iptables* to mitigate large DDoS attacks would saturate the CPUs of the system, which would be occupied discarding traffic rather then

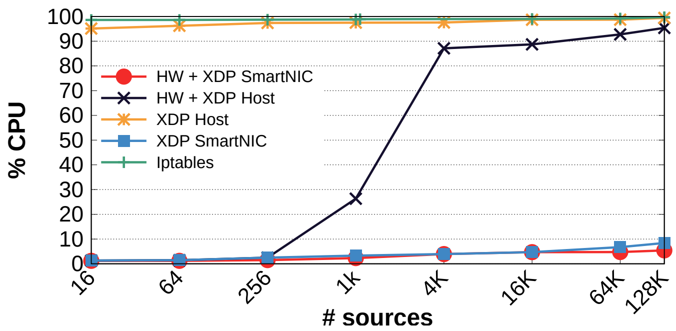


Fig. 3: CPU usage of the different mitigation approaches under a simulated DDoS attack (uniform distribution).

executing the target services.

2) Host-based mitigation: Compared to *iptables*, XDP intercepts packets at a lower level of the stack, right after the NIC driver. This test runs the entire mitigation pipeline in XDP without any help from the SmartNIC, which simply redirects all the packets to the host where the XDP program is triggered. The dropping efficiency of XDP is much higher than *iptables*, being able to discard ~26Mpps up to 1K sources, and still ~10Mpps with 128K attackers, using all CPU cores of the target machine[3]. This performance degradation is due to the eBPF map used (`BPF_HASH`), in which the lookup time, needed to match the IP source of the current packet against the blacklist, is influenced by the total number of map entries.

3) SmartNIC-based mitigation: In this case the mitigation pipeline is executed entirely on the SmartNIC. We performed a first test where the attack is mitigated only through an XDP filtering program in the SmartNIC CPU, without any help from the hardware filter. Results shown in Figures 2a and 2b confirm a performance degradation compared to the host-based mitigation due to the slower CPU of the NIC, balanced by the fact that we do not consume any CPU cycles in the host (Figure 3), hence leaving room for other applications.

A second test exploits a mixture of hardware filtering and XDP-based software filtering in the card. Results demonstrate that for relatively small attack sources (less than 512), the dropping rate is equal to the maximum achievable rate (37.2Mpps); in fact, the first K rules (where K=512 in our card) are inserted in the SmartNIC hardware tables, causing all the packets to be dropped at line rate. However, when dealing with larger attacks (greater than 1K), the dropping rate immediately decreases, since an increasing number of entries stay outside the SmartNIC hardware tables; as a consequence, the dropping rate is influenced by the performance of the XDP program running in the SmartNIC CPU. This approach may be reasonable when the DDoS attack rate does not exceed the maximum achievable dropping rate in the SmartNIC CPU, which in our case is approximately 15Mpps; handling more massive attacks will cause the SmartNIC to drop packets without processing, with an higher chances to drop also legitimate traffic, as highlighted in Section V-C.

[3]In our case, the limiting factor is our Intel Xeon E3-1245 CPU, which is able to drop around 10Mpps within a single core, as opposed to other (more powerful) CPUs that are able to achieve higher rates (e.g., 24Mpps [11]).

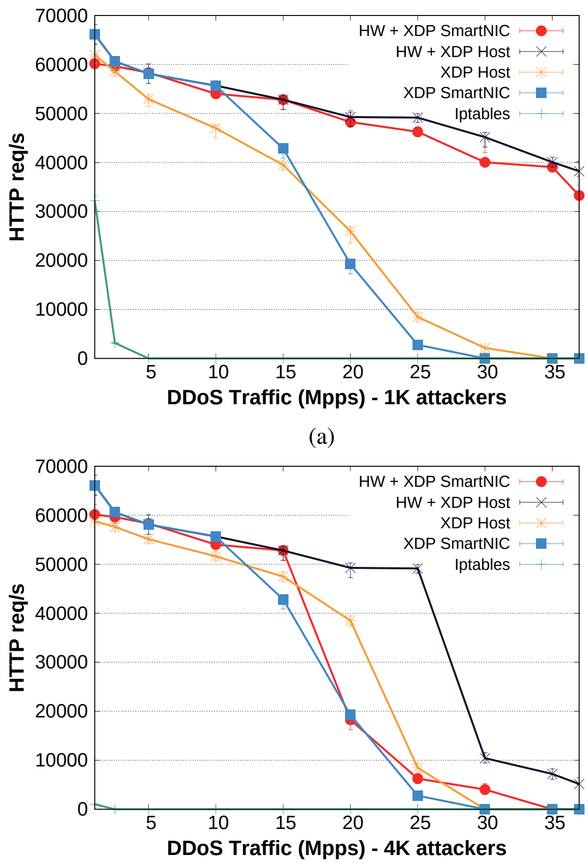
4) Hybrid (NIC Hardware Tables + XDP Host): In this case the offloading algorithm splits the mitigation pipeline between the SmartNIC hardware tables and the XDP *filtering* program running in the host. We notice that for large attacks, the dropping rate is considerably higher than the *HW + XDP SmartNIC* case, thanks to the higher performance of the host CPU compared to the SmartNIC one. Although hardware filtering is available also on some "traditional" NICs (e.g., Intel with Flow Director), we were unable to implement the hybrid approach in them because of the unavailability of hardware counters to measure the dropped packets for each source, which are required by our algorithm; however, we cannot exclude that other mitigation algorithms can leverage the hardware speed-up provided by the above cards as well.

5) Final considerations: Figures 2a and 2b confirm a clear advantage of the hardware offloading, which is even more evident depending on the distribution of the traffic. For instance, in the second scenario (Figure 2b, with some sources generating more traffic than others) we can reach even higher dropping performance, thanks to the offloading algorithm that places the top-K malicious talkers in the SmartNIC, resulting in more traffic dropped in hardware. Also the CPU consumption shown in Figure 3 confirms the clear advantage of the offloading, particularly when most of the traffic is handled by the hardware of the SmartNIC, hence avoiding the host CPU to take care of the above portion of malicious traffic. It is worth noticing that the case where a server has to cope with a limited number of malicious sources may be rather common, as the incoming traffic in datacenters may be balanced across multiple servers (backends), each one being asked to handle a portion of the connections and, hence, also a subset of the current attackers.

C. Effect on legitimate traffic

This test evaluates the capability of the system to perform useful work (e.g., serve web pages) even in presence of a DDoS attack. We generate 64Bytes UDP packets towards the server simulating different attack rates and number of attackers, while a `weighttp` client generates 1M HTTP requests (using 200 concurrent clients) towards the `nginx` server running on the target device. The capability of the server to perform real work is reported by the number of successfully completed requests/s, with a timeout of 5 seconds, varying the rate of DDoS traffic.

Results, depicted in Figures 4a and 4b show the performance with 1K and 4K attackers respectively. In the first case, both *hardware-based* solutions reach the same number of connection/s, since almost all entries are dropped by the hardware, leaving the host's CPU free to perform real work. The same behavior can be noticed when the mitigation is performed entirely on the SmartNIC CPU; in this case, the host's CPU is underused, achieving the maximum number of HTTP requests/s that the DUT is able to handle. However, the performance immediately drop when the attack rate exceeds 15Mpps, which is the maximum rate that the SmartNIC CPU sustain; in such scenario, NIC queues become rapidly full, hence dropping packets without going through the mitigation



(a)

(b)

Fig. 4: Number of successfully completed HTTP requests/s under different load rates of a DDoS attack carried out by *(a)* 1K attackers and *(b)* 4K attackers.

pipeline and increasing the chance to drop also legitimate traffic. With respect to the *XDP Host* mitigation, we notice that the number of connections/s is initially lower, in presence of small attack rates, compared to the *SmartNIC-based* solution, since the host's CPU has to handle the HTTP requests and, at the same time, execute the XDP program. However, when the rate of the attack grows, it will continue to handle an adequate number of connections/s until 25Mpps, which is the maximum rate that the host XDP program is able to handle. Finally, *iptables*-based mitigation results unfeasible with large attack sources because of its very poor processing efficiency, severely impacting on the capability of the server to handle the legitimate traffic.

The same analysis is valid for larger attacks (e.g., 4K sources); the main difference here is that the *HW + XDP Host* solution performs significantly better in this case, thanks to the higher processing capabilities of the host's CPU compared to the SmartNIC ones.

VI. Related work

The advantages of using XDP to filter packets at high rates have been largely discussed and demonstrated [29], [30]; several companies (e.g., Facebook, Cloudflare) have integrated XDP in their data center networks to protect end hosts from unwanted traffic, given the enormous benefits from both filtering performance and low resource consumption. In particular, in [31] Cloudflare presented a DDoS mitigation architecture that was initially based on kernel bypass, to overcome the performance limitations of iptables, and classical BPF to filter packets in userspace. However they shifted soon to an XDP-based architecture called *L4Drop* [32] that performs packet sampling and dropping within an XDP program itself. Our approach is slightly different; we use an XDP program to extract the relevant packet headers from all the received traffic, instead of sending the entire samples to the userspace detection application and we consider simpler filtering rules, which are needed to deal with the SmartNIC hardware limitations. Finally, we consider in our architecture the use of SmartNICs to improve the packet processing, which introduces additional complexity (e.g., select rules to offload), which are not needed in a host-based solution. In this direction, [33] analyzed and proposed a hybrid architecture that use SmartNIC to improve VNFs processing capabilities; however, to the best of our knowledge, this work is the first that analyzes and proposes a complete hardware/software architecture for the DDoS mitigation use case.

VII. Conclusions

Given the sheer increase in the amount of traffic handled by modern datacenters, SmartNICs represent a promising solution to offload part of the network processing to dedicated (and possibly more optimized) components. This paper presents an analysis of the various approaches that could be adopted to introduce SmartNICs in server-based data plane processing, assessing the achievable results in particular for the DDoS mitigation use case under different alternatives. In this respect, the paper describe a solution that combines SmartNICs with other recent technologies such as eBPF/XDP to handle large amounts of traffic and attackers. The key aspect of our solution is the adaptive hardware offloading mechanism, which partitions the attacking sources to be filtered among SmartNIC and/or host, smartly delegating the filtering of the most aggressive DDoS sources to former.

According to our experiments, the best approach is a combination of hardware filtering on the SmartNIC and XDP software filtering on the host, which results more efficient in terms of dropping rate and CPU usage. In fact, running part of the filtering pipeline on the SmartNIC CPU would bring to inferior dropping performance due to its slower CPU, resulting in a lower capability to cope with large and massive DDoS attacks.

Our findings suggest that current SmartNICs can help mitigating the network load on congested servers, but may not represent a turn-key solution. For instance, an effective SmartNIC-based solution for DDoS attacks may require the presence of a DDoS-aware load balancer that distributes incoming datacenter traffic in a way to reduce the amount of attackers landing on each server, whose number should be compatible with the size of the hardware tables of the SmartNIC. Otherwise, the solution may require the software running on the SmartNICs to cooperate with other components running on the host, reducing the effectiveness of the solution in terms of saved resources in the servers.

VIII. Acknowledgement

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence), www.decenter-project.eu.

References

[1] H. Ballani, P. Costa, C. Gkantsidis, M. P. Grosvenor, T. Karagiannis, L. Koromilas, and G. O'Shea, "Enabling end-host network functions," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 493–507. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787493

[2] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving sdn," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 85–90. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342459

[3] Y. Li, D. Wei, X. Chen, Z. Song, R. Wu, Y. Li, X. Jin, and W. Xu, "Dumbnet: A smart data center network fabric with dumb switches," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 9:1–9:13. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190531

[4] T. Karagiannis, R. Mortier, and A. Rowstron, "Network exception handlers: Host-network control in enterprise networks," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 123–134. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402973

[5] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer." in *Hotnets*, 2009.

[6] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding pcie performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: ACM, 2018, pp. 327–341. [Online]. Available: http://doi.acm.org/10.1145/3230543.3230560

[7] DPDK. (2018, jun) Data plane development kit. [Online]. Available: https://www.dpdk.org/ [Accessed: 2019-03-17]

[8] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 101–112.

[9] C. Authors. (2018, jul) Bpf and xdp reference guide. [Online]. Available: https://cilium.readthedocs.io/en/latest/bpf/ [Accessed: 2019-03-17]

[10] M. Fleming. (2017, dec) A thorough introduction to ebpf. [Online]. Available: https://lwn.net/Articles/740157/ [Accessed: 2019-03-17]

[11] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: ACM, 2018, pp. 54–66. [Online]. Available: http://doi.acm.org/10.1145/3281411.3281443

[12] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure accelerated networking: Smartnics in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, 2018, pp. 51–66. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/firestone

[13] A. Caulfield, P. Costa, and M. Ghobadi, "Beyond smartnics: Towards a fully programmable cloud."

[14] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098824

[15] G. Siracusano and R. Bifulco, "Is it a smartnic or a key-value store?: Both!" in *Proceedings of the SIGCOMM Posters and Demos*, ser. SIGCOMM Posters and Demos '17. New York, NY, USA: ACM, 2017, pp. 138–140. [Online]. Available: http://doi.acm.org/10.1145/3123878.3132014

[16] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, and G. Siracusano, "Flowblaze: Stateful packet processing in hardware," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, 2019, pp. 531–548. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/pontarelli

[17] Y. G. Moon, I. Park, S. Lee, and K. S. Park, "Accelerating flow processing middleboxes with programmable nics," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, ser. APSys '18. New York, NY, USA: ACM, 2018, pp. 14:1–14:3. [Online]. Available: http://doi.acm.org/10.1145/3265723.3265744

[18] Arbor Networks. Worldwide infrastructure security report. [Online]. Available: https://pages.arbornetworks.com/rs/082-KNA-087/images/13th_Worldwide_Infrastructure_Security_Report.pdf [Accessed: 2019-03-17]

[19] A. Srivastava, B. B. Gupta, A. Tyagi, A. Sharma, and A. Mishra, "A recent survey on ddos attacks and defense mechanisms," in *Advances in Parallel Distributed Computing*, D. Nagamalai, E. Renault, and M. Dhanuskodi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 570–580.

[20] E. Alomari, S. Manickam, B. Gupta, S. Karuppayah, and R. Alfaris, "Botnet-based distributed denial of service (ddos) attacks on web servers: classification and art," *arXiv preprint arXiv:1208.0403*, 2012.

[21] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," in *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, ser. USENIX'93. Berkeley, CA, USA: USENIX Association, 1993, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267303.1267305

[22] N. Tausanovitch. (2016, sept) What makes a nic a smartnic, and why is it needed? [Online]. Available: https://www.netronome.com/blog/what-makes-a-nic-a-smartnic-and-why-is-it-needed/ [Accessed: 2019-03-17]

[23] ntop. Pf_ring zc (zero copy). [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/ [Accessed: 2019-03-17]

[24] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486011

[25] P. Kamboj, M. C. Trivedi, V. K. Yadav, and V. K. Singh, "Detection techniques of ddos attacks: A survey," in *2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON)*. IEEE, 2017, pp. 675–679.

[26] S. Behal and K. Kumar, "Detection of ddos attacks and flash events using novel information theory metrics," *Computer Networks*, vol. 116, pp. 96–110, 2017.

[27] ——, "Detection of ddos attacks and flash events using information theory metrics–an empirical investigation," *Computer Communications*, vol. 103, pp. 18–28, 2017.

[28] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "An empirical evaluation of information metrics for low-rate and high-rate ddos attack detection," *Pattern Recognition Letters*, vol. 51, pp. 1–7, 2015.

[29] B. Blanco and Y. Lu. (2016, oct) Leveraging xdp for programmable, high performance data path in openstack. [Online]. Available: https://www.openstack.org/videos/barcelona-2016/leveraging-express-data-path-xdp-for-programmable-high-performance-data-path-in-openstack [Accessed: 2019-03-17]

[30] H. Zhou, Nikita, and M. Lau. (2017, apr) Xdp production usage: Ddos protection and l4lb. [Online]. Available: https://www.netdevconf.org/2.1/slides/apr6/zhou-netdev-xdp-2017.pdf [Accessed: 2019-03-17]

[31] G. Bertin, "Xdp in practice: integrating xdp into our ddos mitigation pipeline," in *Technical Conference on Linux Networking, Netdev*, 2017.

[32] A. Fabre. L4drop: Xdp ddos mitigations. [Online]. Available: https://blog.cloudflare.com/l4drop-xdp-ebpf-based-ddos-mitigations/ [Accessed: 2019-03-17]

[33] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. V. Lakshman, "Uno: Uniflying host and smart nic offload for flexible packet processing," in *Proceedings of the 2017 Symposium on Cloud*

Computing, ser. SoCC '17. New York, NY, USA: ACM, 2017, pp. 506–519. [Online]. Available: http://doi.acm.org/10.1145/3127479.3132252

Sebastiano Miano is pursuing his Ph.D. degree at Politecnico di Torino, Italy, where he received his Master's degree in Computer Engineering in 2015. His research interests include programmable data planes, software defined networking and high-speed network function virtualizations.

Roberto Doriguzzi-Corin is a researcher at Fondazione Bruno Kessler in Trento, Italy. He received his M.Sc. degree in Mathematics in 1996 from the University of Trento and he is currently pursuing a Ph.D. degreee at the University of Bologna, Italy. His main research interests focus on network softwarisation, network security and Linux embedded systems.

Fulvio Risso received the M.Sc. (1995) and Ph.D. (2000) in computer engineering from Politecnico di Torino, Italy. He is currently Associate Professor at the same University. His research interests focus on high-speed and flexible network processing, edge/-fog computing, software-defined networks, network functions virtualization. He has co-authored more than 100 scientific papers.

Domenico Siracusa is the head of the RiSING research unit at FBK CREATE-NET. He received the M.Sc. in Telecommunication Engineering (2008) and the Ph.D. in Information Technology (2012) from Politecnico di Milano. His research interests include SDN/NFV, cloud and fog computing, security and robustness. Domenico has co-authored more than 80 scientific papers.

Raffaele Sommese is a Ph.D. Candidate at the University of Twente. He received his M.Sc. in Computer Engineering from Politecnico di Torino in 2018. His research interests focus on DNS DDoS Security, programmable high-speed dataplanes, SmartNIC technologies, and network measurements.