Cerberus: Enabling Efficient and Effective In-Network Monitoring on Programmable Switches

Huancheng Zhou, Guofei Gu
SUCCESS Lab, Texas A&M University

***Abstract*—With the increasing volume of network traffic and the emergence of new types of attacks, traditional network monitoring is facing significant challenges in ensuring network security and performance. In-network monitoring (INM) systems based on programmable switches, e.g., P4-based INM systems, have emerged as a more promising approach for high-performance and real-time network monitoring. However, existing P4-based INM systems have resource limitations in handling diverse and high-volume INM tasks such as multi-vector DDoS defenses. Worse still, attackers may try to dynamically change attack vectors to disrupt inadaptable systems and even lead to denial-of-service (DoS) attacks against INM.**

To address these challenges, we present Cerberus, an efficient and effective in-network security monitoring system. To support various INM tasks, we abstract them into key-feature (K-F) pairs and design a novel *memory slicing* mechanism to share memory among multiple K-F pairs. To handle high-volume traffic, we propose a new co-monitoring mechanism that complements the data and control planes, thereby greatly enhancing the efficiency of Cerberus. To adapt to changing network conditions, we design a new resource manager that dynamically reallocates resources for INM tasks and adjusts loads for the data and control planes without interrupting running services. We design a series of INM modules, including DDoS defenses, and develop a prototype of Cerberus. We conduct extensive evaluations to demonstrate that Cerberus can enhance the concurrency and capacity of programmable switches by an order of magnitude. Moreover, Cerberus is more adaptable in handling various INM tasks.

1. Introduction

Traditional network monitoring has been used for a long time to ensure network security and performance. It mainly involves analyzing network traffic at different locations, such as routers and centralized servers, to identify potential threats or performance issues [1]–[5]. However, coping with diverse and high-volume traffic is becoming increasingly challenging, particularly with the surge of attack traffic such as massive DDoS traffic [6]–[12]. In contrast, in-network monitoring (INM) involves analyzing network activities *within* the network infrastructures, making it a more efficient and effective approach to realizing a series of real-time network functions such as load balancing [13]–[15], distributed denial-of-service (DDoS) defenses [16]–[23], and so on. In particular, recent programmable switch ASICs have emerged as a promising approach to facilitate INM systems, e.g., P4-based INM systems [24]–[34].

As shown in Figure 1, a programmable switch ASIC enables programmers to design highly efficient and customized packet processing pipelines. This makes them suitable for collecting, maintaining, and further analyzing various states of passing packets based on customized policies. Traditional analyzers in the control plane can delegate most tasks to the data plane and query a few sketches containing concise information. Such a cooperative architecture significantly reduces the overhead of the control plane and the communication load between collectors and analyzers, especially in large-scale network monitoring scenarios.

Despite the promised high performance and flexibility, existing P4-based INM systems can either serve only a very small number of concurrent INM tasks or only handle a certain volume of traffic, making it challenging to defend against diverse and volumetric attacks. For example, P4-based DDoS defense systems [24], [25] only run several types of defenses at the same time. When network administrators detect new attacks, they need to reload new programs to the switches. Unfortunately, this can cause non-negligible downtime (e.g., a few tens of seconds). An advanced attacker can exploit the downtime to disrupt P4-based INM systems and even lead to denial-of-service (DoS) attacks against INM by quickly changing attack vectors.

The bottleneck of current P4-based INM systems is mainly due to the resource limitations of programmable switches. First, to maintain line-rate speed, programmable switches restrict the number of memory accesses per packet, making them infeasible to perform multiple INM tasks, which access memory to store stateful information. Second, programmable switches have limited on-chip memory (e.g., SRAM). A single INM task (e.g., SYN flood defender) that deals with a large amount of traffic (e.g., all TCP flows) can consume a significant portion of available memory, leading to insufficient memory for running other INM tasks. Some researchers have attempted to alleviate the bottleneck through sophisticated designs [25], [29], [35]. However, these designs may not be effective when faced with tricky attackers who deliberately generate a large number of abnormal flows or frequently change attack/packet types.

To address the bottleneck, we propose Cerberus, an efficient, effective, and high-performance in-network security monitoring system against diverse, high-volume, and

dynamic attacks. First, we observe that there are various network monitoring tasks. Thus, we abstract each INM task into one or multiple key-feature (K-F) pairs, so we can merge K-F pairs involving similar features. We further propose a novel *memory slicing* mechanism to share memory among multiple K-F pairs. Therefore, multiple INM tasks can access different states with a small amount of memory access units (i.e., arithmetic logic units). Second, to support INM tasks with high memory requirements, we design a new co-monitoring mechanism to complement the data and control planes. We utilize the on-chip memory in the data plane to store the most frequently updated bits, and the large memory in the control plane to store the less frequently updated bits. With appropriate collaboration, we enhance efficiency by an order of magnitude with minimal impact on performance. Third, to adapt to the changing network conditions and dynamic attacks, we design a new resource manager to dynamically reallocate resources for different K-F pairs at runtime. Moreover, the resource manager can adjust the workload distribution of the data and control planes. Therefore, Cerberus can not only provide uninterrupted service but also consistently maintain high performance.

To sum up, we make the following contributions:

- To support diverse INM tasks effectively, we abstract INM tasks into K-F pairs and design a novel memory slicing mechanism.
- To handle high-volume traffic, we design a new co-monitoring mechanism that enhances the efficiency of P4-based INM systems.
- To meet dynamic requirements, we develop a new resource management mechanism to reallocate the resource and workload distribution without interrupting running services.
- We implement a prototype of Cerberus (code release at [36]) and conduct extensive evaluations, which demonstrate that Cerberus is more effective, efficient, and adaptable in handling a series of INM tasks.

2. Background and Motivation

Nowadays, network operators are concerned with a lot of types of network traffic states, such as flow speed, number of connections, and even inter-packet delay. With detailed states, network operators can manage networks efficiently, detect anomalies precisely, or mitigate attacks timely. A network monitoring system usually needs to collect network states from forwarding devices. Since forwarding devices such as routers only provide simple functionalities, passing packets need to be mirrored to external systems with sophisticated applications for further analysis [1]–[5]. However, the increasing volume of network traffic makes it more and more costly for external systems to pull a precise network view. As a result, most systems have to set a low sampling rate to reduce overhead [17], [37]–[40]. Recently, programmable switches provide opportunities for collecting, storing, and analyzing states locally, i.e., in-network monitoring (INM). Compared to traditional network monitoring, INM systems based on programmable switches (e.g., P4-based INM systems) are high-performance and flexible.

2.1. Programmable Switch ASICs

Today's data centers and networks have higher throughput and diverse demands, so programmable switch ASICs become competitive compared to fixed-function switch ASICs. With domain-specific languages such as P4 [41], programmers can define an efficient and customized packet processing pipeline. Without sacrificing performance, programmable switch ASICs support a lot of new or customized network functions [13], [24]–[27], [42]–[46].

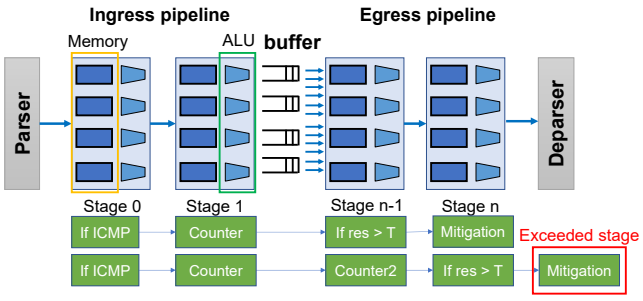


Figure 1: Protocol Independent Switch Architecture.

Pipeline of programmable switch ASICs. As shown in Figure 1, programmers can define the packet processing pipeline as they need. More specifically, programmers first allocate resources (e.g., memory, ALUs) in each **stage**, which own individual resources. For example, allocating memory to stateful memory object (**registers**) to cache runtime states. Second, programmers specify the processing logic of packets (e.g., add headers) in each stage, e.g., defining the match-action **tables** and table **actions**. Besides, programmers can customize the register actions to determine the operations of registers. One important thing is that all stages perform the defined operations at the same CPU cycle. Thus, most resources (e.g., register and table) can only be accessed by their owner stage. Otherwise, there will be race conditions. Moreover, as the resources of stages, all actions must be as simple as possible to execute within one stage (one CPU cycle). To maintain line rate speed, most programmable switches only support simple operations. For example, addition, subtraction, and logical operations (e.g., and/or).

2.2. Challenges in P4-based INM systems

Although programmable switches provide a high-performance and customized pipeline to realize in-network monitoring, there are several challenges due to their features.
C1. Limited resource. Some resources of programmable switches are relatively limited. For example, a programmable switch pipeline usually has a few tens of stages to ensure high speed. Each stage only owns a few arithmetic logic units (ALUs) for memory access. Therefore, the

number of INM tasks is restricted. Second, a programmable switch only owns tens of megabytes of on-chip memory (e.g., SRAM), which is insufficient for large-scale and hybrid INM scenarios.

C2. Exclusiveness. Worse still, most resources of programmable switches are exclusive, which means that they can only be accessed by their owners. For example, if an ALU or a register is assigned to a table object, then other tables cannot access it. Even if some registers are idle, they cannot be reassigned to other tables to store other states.

C3. Slow adaption. Given C1 and C2, a straightforward solution is to generate new configuration files and programs when there are new requirements. However, even if we ignore the time of generating a new program, the subsequent recompiling and reinstalling processes may still take a few tens of seconds, resulting in non-negligible downtime. Furthermore, a tricky attacker could exploit this vulnerability and force the programmable switch to keep resetting, causing a denial-of-service (DoS) attack.

2.3. Approximate data structures in INM

Approximate data structures, e.g., sketches, are widely used in INM to reduce the amount of data that needs to be processed and stored. Sketches can quickly provide an approximate estimate of large and real-time data streaming for tasks such as traffic monitoring, anomaly detection, and attack mitigation. Typically, sketches use a compact representation of the data through counters or hash tables that summarize key characteristics of traffic (e.g., packet count). Popular sketches include Bloom Filter (BF) [47], Counting Bloom Filter (CBF) [48], Count-min Sketch (CMS) [49], and so on. Collisions may occur in sketches, but the collision rate is negligible unless it is overfilled. Before using sketches, we need to estimate the amount of data n to be stored approximately. For example, given a BF with the size of m, we can calculate the collision rate using the formula: $p \approx (1-e^{-\frac{kn}{m}})^{\frac{m}{kn}}$, where k is the number of hash functions.

3. Overview

In this section, we first define the scope of our research problem, then we indicate our design goals. Finally, we provide an overview of Cerberus.

3.1. Problem Scope

In-network monitoring (INM). In this paper, we concern with INM on programmable switches, which collect and process various states of passing packets. For example, the number of packets sent by a host, the number of special types of packets, the inter-packet delays of a flow, and so on. We define two types of INM tasks, sketch-based INM and non-sketch-based INM. We mainly focus on sketch-based INM tasks, which leverage diverse sketches to store concise information (e.g., packet count, connection existence). These sketches are widely used due to their lightweight overhead. Unfortunately, this lightweight structure with limited memory can lead to a high collision rate in large-scale monitoring scenarios (e.g., handling hundreds of thousands of flows). Besides, there could be multiple concurrent sketch-based INM tasks, which further reduce the available memory of each INM task. For non-sketch-based INM tasks based on other data structures such as key-value stores, they usually focus on small-scale traffic sets due to high memory cost. Thus, we do not target them in this paper.

Security goals. Our focus is on addressing the limitations of programmable switches from a security perspective. Specifically, traditional INM tasks are often assumed to be well-planned and predictable, with resource allocation based on expected traffic loads. However, in a security scenario such as DDoS defenses, the specific type, volume, and target victims cannot be accurately predicted. Thus, we must be prepared to handle dynamic and adaptive adversaries.

Threat model. Cerberus is deployed on programmable switches within network infrastructures to perform configured INM tasks and collect traffic states. However, the presence of attackers who control botnets may disrupt the availability of protected servers or networks, such as launching denial of service (DoS) attacks. These attackers are well aware that the victims are protected by various INM tasks, prompting them to devise strategies to bypass these defenses. One approach is to launch hybrid attacks, exploiting resource-constrained systems with a partial deployment of INM tasks. Additionally, attackers may rapidly change the postures of their attacks to trigger frequent reconfigurations of programmable switches, leading to disruptive interruptions or DoS attacks. Cerberus aims to mitigate the impact of hybrid and dynamic attacks and safeguard network availability and performance.

3.2. INM Abstraction

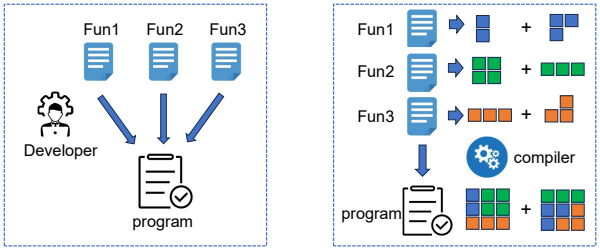


Figure 2: The abstraction of INM tasks.

Traditionally, developers face the challenge of manually composing multiple INM tasks (functions), which is a time-consuming and error-prone process due to limited and exclusive resources. Function-level composition often requires careful resource allocation to avoid excess usage or race conditions. To simplify the programming process, we introduce a resource-level composition approach. That is, each INM task is represented by several key-feature (K-F)

pairs. A K-F pair means recording a specific feature corresponding to a given key, which occupies a part of resources. This approach simplifies the programmer's role, as they only need to focus on selecting INM tasks by combining various K-F pairs. Leveraging a modular programming style, our compiler intelligently assesses and automatically combines different K-F pairs, reducing manual effort and enhancing efficiency.

3.3. Design Goals

To meet various requirements of INM tasks, our system should realize the following features:

- **High concurrency**. Each packet stream may have multiple keys, and there could be dozens of features associated with a specific key. In many cases, such as hybrid DDoS defenses, defenders need to define different keys and query multiple features simultaneously. Therefore, Cerberus should be able to deploy and run many INM tasks concurrently.
- **High capacity**. Current networks are capable of handling billions of packets and millions of flows per second. During attacks, the number and volume of flows can remain at a high level for an extended period. Hence, Cerberus must be equipped to handle large-scale INM tasks effectively.
- **Run-time adaptability**. The conditions of a real-world network can change unpredictably due to burst traffic, malicious events, and so on. Different time windows may also require different INM tasks, such as large-scale performance monitoring or multi-vector DDoS defenses. Unfortunately, current programmable switches take a while (e.g., a few tens of seconds) to recompile and reinstall new programs with updated configurations. An advanced attacker may exploit this feature to force programmable switches to keep resetting, leading to a DoS attack. Therefore, Cerberus needs to quickly adapt to different strategies without introducing any downtime.

3.4. Workflow

To achieve these goals, we design Cerberus as shown in Figure 3. Developers first configure the data and control planes by setting K-F pairs. When network traffic passes through switches, the data plane can collect and process the state locally. To perform multiple INM tasks concurrently with constrained resources, we design a memory slicing mechanism, which allows multiple INM tasks to use the same ALUs and registers. In addition, we design a co-monitoring mechanism to store overflowed bits, which are *carry flags* generated during state aggregation. Therefore, the data plane always records the low bits (i.e., least significant bits) and the control plane maintains the remaining bits (i.e., most significant bits). The complete states can be recovered as needed. During this process, a resource manager will automatically allocate resources to functions according to real-time network conditions. For example, when there is a lot of malicious traffic (indicated by the red arrow) passing through the protected network, the running detectors can identify the types of attacks and activate corresponding defense functions. Meanwhile, the data plane will report the events to the control plane. Cerberus also allows manual configuration for strategy adjustment. Network managers can easily notice the reported events and adjust their policies. Finally, most malicious traffic is expected to be filtered.

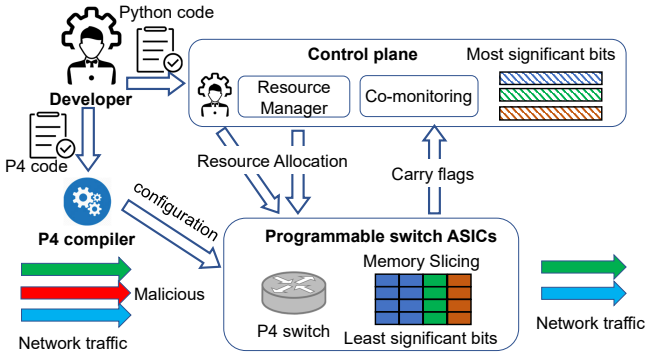


Figure 3: The workflow of Cerberus.

4. Cerberus's Design

4.1. Memory Slicing

One requirement of P4-based INM systems is to support many types of INM tasks concurrently. Generally, there can be tens of keys and hundreds of features for passing traffic (e.g., tens of packet types and hundreds of headers). It is common for a network to monitor tens of K-F pairs concurrently [37], [38]. However, existing P4-based INM systems can hardly support them due to resource limitations.

To address this issue, we propose a memory slicing mechanism to *share the same registers among multiple K-F pairs*. The key idea is to **concatenate** multiple K-F pairs so that different functions can access their K-F pairs in a register at the same time. As shown in Figure 4, a packet may trigger multiple conditions, so the data plane needs to update multiple features in the same register using different actions or parameters. When a packet triggers the preconditions of multiple applications (①), the table outputs the triggering flags (②). Then, a memory slicing table is used to determine the current K-F pairs (③). The multiple K-F pairs are merged to create a concatenated K-F pair (④). Finally, the concatenated K-F pair is used to update multiple features within one register access (⑤).

One challenge of the memory slicing mechanism is to perform different actions with a single ALU. For example, a K-F pair may require incrementing by the length of a packet (e.g., byte counter), while another K-F pair may need to record the latest timestamp of a flow. We observe that most INM tasks mainly involve a few types of operators, such

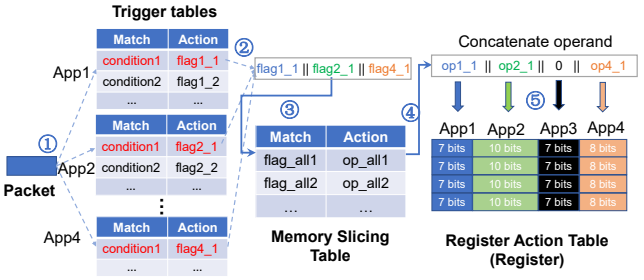


Figure 4: The workflow of the memory slicing mechanism.

as addition, subtraction, and assignment[1]. Although their operands may be different, we can concatenate K-F pairs with the same operators.

Addition/subtraction. Many counters involve addition operators that aggregate the states of all passing packets. For example, a byte counter aggregates the number of bytes of given keys, and a SYN flood detector needs to record the number of SYN packets. In some cases, such as removing an entry in CMS, subtraction is also necessary. To merge multiple addition/subtraction operations, we first collect the operands of triggered K-F pairs in ③. Given the offset of each K-F pair (e.g., app2's offset is 15), we can calculate the concatenated operand. Finally, we execute the register action with the concatenated operand. It is worth noting that we take the inverse of operands for subtractions. For example, to subtract X, we can add inverse(X), where inverse(X) is equal to $2^n - X$ and n is the length of the register.

Assignment. Some K-F pairs involve assignment operators such as inter-packet delay recorder (i.e., recording the timestamp of packets). Cerberus's resource manager first chooses to merge K-F pairs with unconditional assignments. For conditional assignments, the resource manager estimates their frequency and available resources, then decides whether to merge them or split them into distinct registers. For example, a duration byte counter performs addition during a window and is set to 0 (i.e., assignment) at the end of the window. Since it performs only one assignment per window, the assignment frequency is relatively low, and it can be merged with other addition operators.

Carry flag. When we add multiple features concurrently, the carry bit of features may be lost or poison other features. Take Figure 4 as an example, when the feature of App1 is larger than 2^7, its carry bit (8-th bit) will be lost. Besides, when we increment the feature of App4 to a value larger than 2^8, the carry bit will be added in the 9-th bit, which belongs to App3. To achieve state isolation, we tag the highest bit of each feature as a secure bit (*carry flag*). Whenever the data plane detects the carry flag of any slice is set to 1, it subtracts the carry flag by recirculating a mirror packet and re-accessing the same item. In a nutshell, we constrain each feature to its own slice, preventing features from crossing their boundary and poisoning other features. To avoid data loss, the data plane uploads the carry flags to the control plane, as shown in the next subsection "Co-monitoring".

1. To maintain high performance, programmable switches only allow simple operators.

4.2. Co-monitoring

Another requirement of P4-based INM systems is to handle high-volume traffic. Unfortunately, due to limited resources and static resource allocation, it is infeasible to configure resources properly for multiple K-F pairs under changing network conditions. More specifically, the consumed space of a register is determined by its size and length, which are fixed once the program is run. In most cases, developers have to define a "safe" length for registers to prevent overflow. This typically means setting the length of registers to match the length of the maximal value. However, having long register lengths often imposes restrictions on the size of the registers.

We observe that most items in registers are much less than the maximum value, which is intuitive due to the fact that most network flows are mice flows. For example, a real-world online trace dataset [50] contains hundreds of thousands of IP addresses. Their average rate is about 13 packets per second (pps), but only 4.6% of IP addresses exceed the average rate. Therefore, for most IP addresses or flows, a short length is enough for a packet counter. To support various lengths in the same register, we propose a co-monitoring mechanism. The key idea is that the data plane stores the least significant bits that are updated most frequently while the control plane only stores the most significant bits that are updated least frequently. As a result, we avoid introducing heavy overhead to the control plane.

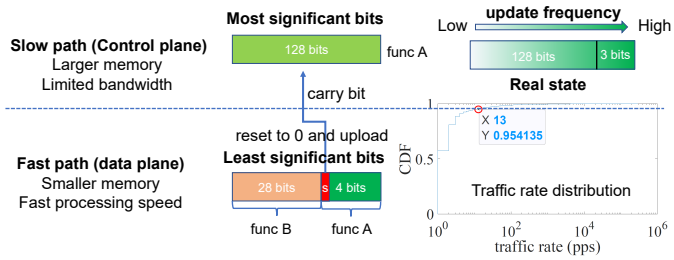


Figure 5: Co-monitoring Mechanism

As shown in Fig. 5, when the highest bit (bit s) of feature A is set to 1 (carry flag = 1), the data plane recirculates a mirror packet to clear the carry flag. Subsequently, the mirror packet is uploaded to the control plane to retain the carry flag. The carry flag is added to the lowest bit for the same K-F pair. By doing so, we improve the available length of each state. We assume L_D and L_C are the lengths of K-F pair in the data and control planes, respectively. Then the actual length of the K-F pair is $L_D + L_C$. In other words, the available memory for this K-F pair is $\frac{L_D+L_C}{L_D}$ times the original memory.

Overhead estimation. An obvious question of the co-monitoring mechanism is that the communication overhead between the data and control planes can be high. Here, we formally estimate the overhead. Assume that there are

M individual flows and N slices to store N types of features. The length of the i-th slice in the data plane is L_{D_i} $(0 \leq i \leq N-1)$. Within a refresh cycle c, the final value of the i-th feature of j-th flow is $V_{i,j}$ $(0 \leq j \leq M-1)$. We define a metric called hit per second (HPS), then we can calculate the HPS of the i-th feature and the j-th flow by Eq. 2, where $ReLU$ is a rectified linear activation function that ignores small values, and $U_{i,j}$ is the incremental value of the i-th state of the j-th flow[2]. For the i-th slice, we can get HPS_i using Eq. 3. Furthermore, we can calculate the recirculation times per second (RTPS) and consuming bandwidth (CB) using Eq. 4 and Eq. 5, where m is the size of each uploaded packet.

$$ReLU(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{otherwise} \end{cases} \tag{1}$$

$$HPS_{i,j} = ReLU(\lfloor V_{i,j} / 2^{L_{D_i}-1} \rfloor)/cU_{i,j} \tag{2}$$

$$HPS_i = \sum_{j=0}^{M-1} HPS_{i,j} \tag{3}$$

$$RTPS = \sum_{i=0}^{N-1} HPS_i \tag{4}$$

$$CB = RTPS \times m \tag{5}$$

To avoid overwhelming the control plane, we should ensure that CB is less than the predefined thresholds. Obviously, a longer slice (larger L_{D_i}) can reduce HPS and CB, but it also occupies more memory in the data plane. By measuring the real-time HPS and CB, the control plane can adjust the length of each slice to keep CB less than the predefined thresholds without consuming unnecessary space in the data plane.

4.3. Resource Manager

As mentioned in §3.3, developers usually need to reconfigure programmable switches by reinstalling programs, which leads to non-negligible downtime. In this part, we show how Cerberus's resource manager can adapt to changing network conditions.

Pipeline Optimization. When developers assign multiple K-F pairs, the resource manager will analyze their triggering conditions and involved actions to arrange them properly.

First, the resource manager examines the mutual exclusivity among all K-F pairs. Since mutually exclusive K-F pairs will not be triggered at the same time, the resource manager can allocate them to the same register with different index offsets. For example, App1 can access the range from $[0, size_1)$, App2 can access the range from $[offset_2, offset_2 + size_2)$, and so on.

2. For example, the incremental value of a packet counter is 1 for any flows, while the incremental value of a byte counter is decided by the packet length of each flow.

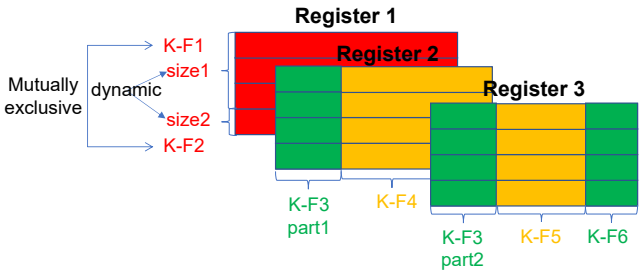


Figure 6: Resource allocation strategy.

Second, the resource manager tries to allocate K-F pairs with the same key to the same registers. The resource manager labels K-F pairs with three levels, i.e., green, yellow, and red. K-F pairs involving compare operations in register actions are labeled as red. K-F pairs involving inconstant operands or multiple operators are labeled as yellow. K-F pairs involving constant operands and a single operator are labeled as green. As shown in Figure 6, the resource manager first allocates distinct registers for K-F pairs with red labels[3]. Then, the resource manager allocates registers for K-F pairs with yellow labels. When there are not enough registers, the resource manager will share registers among them. Finally, the K-F pairs with green labels are unrestricted to share registers with other K-F pairs.

Adaptable Memory Space. An application may want to use its memory in different ways under different situations. For example, when the application handles a few connections, it tends to perform high-precision monitoring, so it declares a long register (e.g., width = 32). However, when there are more network connections, it may want to declare a large register (e.g., size = 262144). Since we cannot predict the network conditions, it seems impossible to define a "perfect" register to fit different cases.

Fortunately, we can dynamically change the length and position of slices. As shown in Figure 7, we can shorten the length of the original state state1 from 32 bits to 8 bits and create three mirrored slices for the same state. As a result, we use the same register to meet different types of requirements. The cost is that the length of the original states might be too short to record large values. However, we can upload the high bits to the control plane by using the co-monitoring mechanism. Besides, we can reduce the precision of INM tasks, so they will use shorter states[4]. Finally, we indicate that the demanding INM tasks can also temporarily "borrow" memory from other idle INM tasks. That is, we can reduce the length or the size of other slices and assign them to the demanding INM tasks.

Filtration. When Cerberus performs the co-monitoring mechanism, there could be some elephant flows with large

3. If there are not enough registers, the resource manager generates an alert asking the developers to adjust the programs (e.g., refactoring codes to use compare operations outside registers).

4. For non-increasing states such as saving the timestamp of packets, we can reduce the precision such as saving an 8-bit timestamp instead of 32-bit timestamp [28].

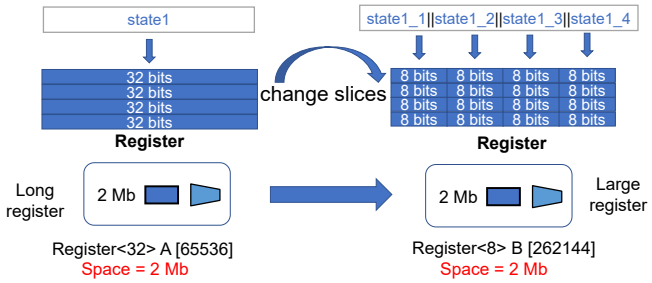


Figure 7: Adaptable register

states. In the worst case, the elephant flows may trigger multiple INM tasks and occupy multiple slices requiring a length that is longer than the available length. Given the predefined threshold T_{CB}, there is no solution if $\sum_{i=0}^{N-1} max(HPS_i) \times m > T_{CB}$, where $max(HPS_i)$ is the maximal value of given flow sets for the $i-$ th state.

To address this issue, we propose a filtration method that involves segregating elephant flows from the remaining flows. Specifically, by analyzing the $HPS_{i,j}$ values for the i-th slice and j-th flow, the control plane can easily identify the elephant flows that trigger recirculation most frequently. Subsequently, the control plane can establish flow rules to store these elephant flows in a separate region (elephant region) with longer slices and a smaller size. As depicted in Figure 8, we can utilize the remaining space to create this elephant region. Since only the top-k (e.g., 8192) heaviest flows need to be stored, a smaller space is sufficient. Given resource limitations, an attacker cannot generate an excessive number of elephant flows simultaneously. In the worst cases, we can also expand the elephant region by reallocating space from the main region using virtual addressing.

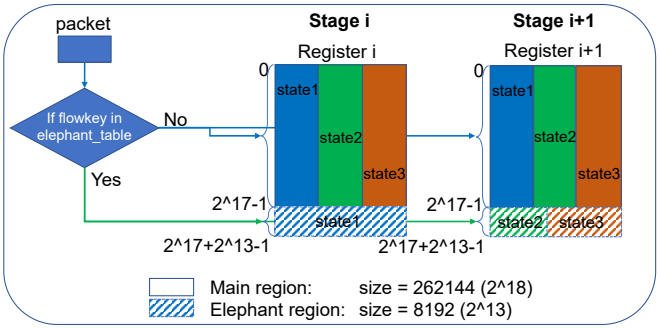


Figure 8: Special region for elephant flows.

Dynamic eviction. As the network conditions change, the top-k heaviest flows can also change. Moreover, an advanced attacker may intentionally change flow speeds to evade the capture of the elephant region. Therefore, we need to dynamically evict stale elephant flows from the elephant region. Typically, attackers need a few seconds to reach the k-th highest value. Meanwhile, the values (e.g., speed) of old flows will be cleared accordingly. Therefore, the control plane can periodically query the values in the elephant region and proactively kicks out the flows with lower values. Since the instructions from the control plane are not atomic, we first disable the match action rules so that the evicted flows will be recorded in the main region. Then the control plane reads and clears their states in the elephant region. Finally, the control plane saves the higher bits locally and sends the remaining bits to the data plane. In general, the change of top-k heaviest flows may not be significant. Otherwise, the dramatic change in speed can serve as a useful pattern to detect potential attacks or network anomalies.

5. Implementation

We have implemented a prototype in P4 [41] on Barefoot Tofino [51] switch with P4 Studio [52]. We will open-source our prototype.

Data plane. In regards to the memory slicing mechanism, we utilize a memory slicing table to map various applications to their corresponding slices. After each register access, we check whether the highest bit of any slice is set to 1. If so, the data plane sets the mirror flag to copy the packet. After the packet has been mirrored, it is recirculated to the beginning of the pipeline. When detecting a mirrored packet, the data plane can extract the position of the highest bits, which can then be reset to 0 by subtracting the value embedded in the mirror header. In terms of storage data structures, we use approximate data structures such as BF [47] and CMS [49] for most INM tasks. In comparison to direct mapping, they can store much more items with a controllable collision rate.

Control plane. We implement the control plane program in Python. It mainly receives carry flags from the data plane and dynamically adjusts the length of slices to meet real-time requirements. Besides, the control plane can insert flow rules into the data plane based on the collected states. For instance, the control plane can assign a flow to the elephant region by inserting a flow rule to a trigger table. In cases where INM tasks require the most significant bits, the control plane can directly read local states. For INM tasks requiring full bits of states, the control plane first filters irrelevant items by analyzing the most significant bits. Then, it queries the least significant bits from the data plane.

Programming INM with Cerberus. We design various modules as a library, as shown in Table 1. Developers can input a configuration file containing the required modules in our library. Cerberus's compiler will automatically generate the required codes related to resource allocation. Developers only need to focus on the actions and conditions (e.g., tables and actions), which are simple to compose.

As shown in Table 1, Cerberus supports various keys such as source/destination IP address (srcIP/dstIP), host pair (2-tuple [srcIP | dstIP]), host pair with layer 4 ports (4-tuple [srcIP | dstIP | sport | dport]), and so on. It also provides a wide range of features, including byte count, packet count, existence, and so on. Developers can configure a (srcIP, UDP count) pair to record the number of UDP packets of source IP addresses (UDP flood defense). If the UDP count exceeds a predefined threshold, the corresponding mitigation module is triggered. Similarly, to defend against DNS amplification

TABLE 1: Examples of K-F definition of INM tasks

INM tasks	Key	Feature
ICMP (F1)	2-tuple	ICMP count
Smurf attack (F2)	2-tuple	ICMP count
Coremelt (F3)	2-tuple	Byte count
DNS amplification (F4)	4-tuple	Query existence
UDP flood (F5)	2-tuple	UDP count
DNS flood (F6)	2-tuple	DNS count
NTP amplification (F7)	4-tuple	Query existence
SSDP amplification (F8)	4-tuple	Query existence
Memcached amplification (F9)	4-tuple	Query existence
QUIC amplification (F10)	4-tuple	Query existence
HTTP flood (F11)	4-tuple	HTTP count
Slowloris (F12)	2-tuple	Connection count
SYN flood (F13)	2-tuple	SYN count
ACK flood (F14)	5-tuple	SYN existence
RST/FIN flood (F15)	5-tuple	SYN existence
Rate counter (F16)	2-tuple	Packet count
NetWarden (F17)	5-tuple 5-tuple	Timestamp Inter-packet delay

attacks, developers can also configure a (2-tuple, Query existence) pair. If a source IP address does not send a DNS query (Query existence = 0) to a destination IP address but receives its DNS replies, the DNS replies will either be dropped or rerouted to a honeypot based on predefined policies.

6. Evaluation

In this section, we evaluate Cerberus with respect to four key questions:

- Can Cerberus efficiently handle large-scale INM tasks?
- Can Cerberus efficiently perform dozens of concurrent INM tasks?
- Can Cerberus effectively defend against hybrid and dynamic attacks?
- Does Cerberus introduce a small impact on throughput, available bandwidth, and process latency?
- Is Cerberus robust under various attacks?

Testbed. Our testbed includes two Wedge100BF-32X programmable switches with Tofino chips [51]. We use two servers to generate malicious traffic and background traffic. The maximum volume of malicious traffic is 10Gbps.

Traffic generation. The background traffic is sourced from an online trace dataset [50]. When generating malicious traffic, we utilize some tools such as scapy and hping3. To send packets at a high rate, we first save malicious packets and then replay them by using tcpreplay.

Estimated metrics. When performing classification tasks such as DDoS defenses, we may set certain policies to determine whether a flow is malicious or benign. For instance, we can set thresholds and compare the measured values with them, or we can set a BF and verify the existence of a key. However, a flow can be misclassified due to collisions, which can be estimated by various metrics, including relative error (RE), false positive rate (FPR), false negative rate (FNR), and malicious traffic ratio. Since we have the ground truth, it is easy to get the real values. When calculating RE, we use the equation $RE = \frac{real-measure}{measure}$ with real values ($real$) and measured values ($measure$). When calculating FPR and FNR, we use the equation $FPR = \frac{FP}{TN+FP}$ and $FNR = \frac{FN}{TP+FN}$.

6.1. Cerberus's Capacity

Settings. In this section, we compare Cerberus with a host rate counter that analyzes the behaviors of each host. We employ multiple count-min sketches (CMS) counters to store the rate of each host, which follows the implementation approach used in Ripple [26]. We set three baselines. The first one uses four CMS arrays with a size of 2^{16}, the second one uses four CMS arrays with a size of 2^{17}, and the last one uses four CMS arrays with a size of 2^{18}, respectively. All CMS arrays have a width of 32 bits. Therefore, they totally use 1MB, 2MB, and 4MB memory. For Cerberus, we create four CMS arrays with a size of 2^{16} and a width of 32 bits (1MB memory), but we support the memory-slicing and co-monitoring mechanisms. We replay four real-world datasets (a small-scale, a medium-scale, and two large-scale traffic sets) [50] at different rates (from 100Mbps to 5Gbps). On average, the traffic sets 1, 2, 3, and 4 contain ~8,000, ~30,000, ~43,000, and ~60,000 unique IP addresses, respectively. Since the host rate counter does not involve classification, we estimate the RE of different solutions.

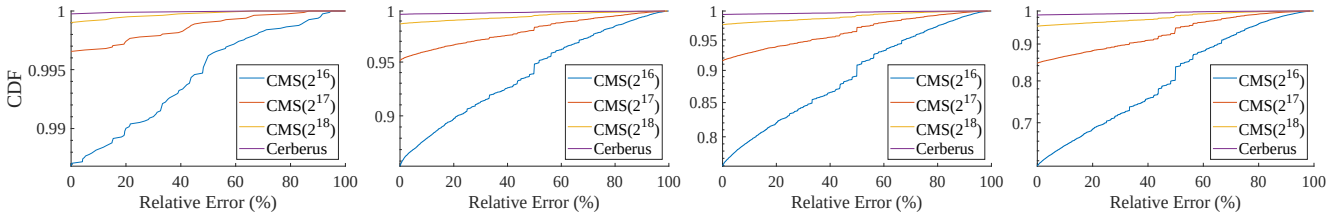
As shown in Figure 9a, all CMS counters achieve low RE (RE of over 98% of hosts is less than 5%) when there are a small number of hosts. However, the RE of baselines using less memory (e.g., 1MB and 2MB) increases greatly when there are more hosts, as shown in Figure 9b, 9c, and 9d. The baseline using 4MB achieves low RE (RE of over 99.3% of hosts is less than 5%), but the cost is using multiple times the memory. On the contrary, Cerberus uses less memory while maintaining the lowest RE in all cases (RE of over 99.3% of hosts is less than 5%). This is because Cerberus assign a short slice for the host rate counter (slice length is 4 bits in our settings). Meanwhile, Cerberus utilizes the co-monitoring mechanism to record the remaining bits of long states. The cost is that the data plane may need to upload some packets to the control plane. However, the estimated results in section 6.4 show the proportion of uploaded packets is negligible and reducible.

Takeaway: Cerberus can enlarge the capacity of a UDP packet rate counter by 8 times[5].

6.2. Cerberus's Concurrency

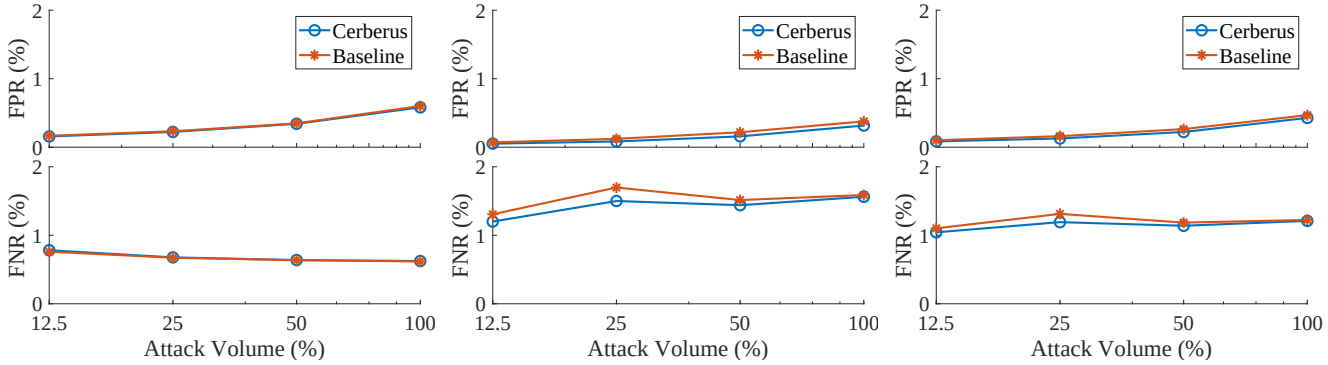
Settings. In this section, we demonstrate that Cerberus can deploy multiple concurrent INM tasks efficiently. To estimate the FPR, we evaluate three scenarios for multi-vector DDoS defenses. The first one contains a UDP flood defense, a Coremelt defense, and a DNS amplification defense. The second one includes a Smurf defense, an HTTP flood defense, an SSDP amplification defense, a Slowloris defense,

5. We only use 1MB memory (width = 4 bits) in the data plane, but the control plane keeps remaining bits (width = 28 bits), so the total available memory is 8MB.



(a) RE on traffic set1 (100Mbps) **(b)** RE on traffic set2 (1Gbps) **(c)** RE on traffic set3 (2Gbps) **(d)** RE on traffic set4 (5Gbps)

Figure 9: Relative error (RE) of different CMS settings on different traffic sets.



(a) Mitigation FPR/FNR on 3 attacks **(b)** Mitigation FPR/FNR on 5 attacks **(c)** Mitigation FPR/FNR on 8 attacks

Figure 10: Multi-vector DDoS defenses, where the maximal attack volume is 10Gbps.

and a QUIC amplification defense. The third one contains the F1 - F8 functions, as shown in Table 1. We utilize real-world datasets [50] to replay background traffic at a speed of 1Gbps. Then we generate malicious traffic with a total speed of 10Gbps. To evaluate the efficiency of Cerberus, we compare it with a baseline that deploys corresponding defenses with individual resources for each function. The baseline solution aligns with the implementation of existing solutions such as Jaqen [25]. In contrast, Cerberus supports the memory slicing mechanism, enabling resource sharing among multiple functions. To demonstrate that Cerberus can identify malicious traffic and benign traffic, we estimate the FPR and FNR of different solutions.

As shown in Figure 10, both Cerberus and the baseline achieve low FPR and FNR. However, Cerberus only uses a few registers and ALUs. This is because Cerberus's memory slicing mechanism can share memory and ALUs among multiple defenses. Moreover, when there are more attacks or a higher volume of traffic, Cerberus's resource manager can adjust the workload distribution of the data and control planes. Therefore, the control plane can store a part of bits that are updated least frequently. We further compare the resource usage of the baseline and Cerberus. As shown in Figure 11, the resource usage of the baseline increases as the variety of INM tasks increases. When we deploy three types of INM tasks, the stage resource usage exceeds the limit, so we have to use more programmable switches to deploy them. When we add more INM tasks (e.g., F1-8), other resources such as SRAM or ALUs also exceed the available value[6]. On the contrary, Cerberus (8 types) occupies much less resources with the help of the memory slicing and co-monitoring mechanism.

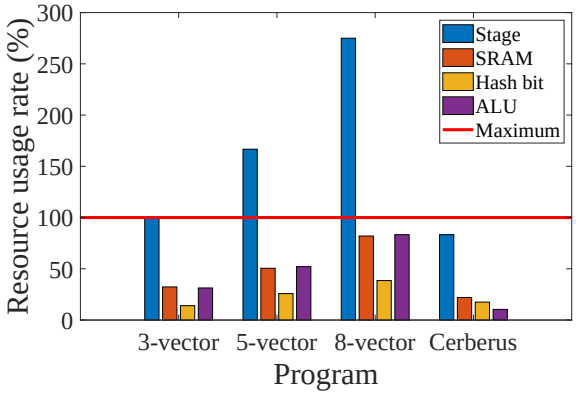
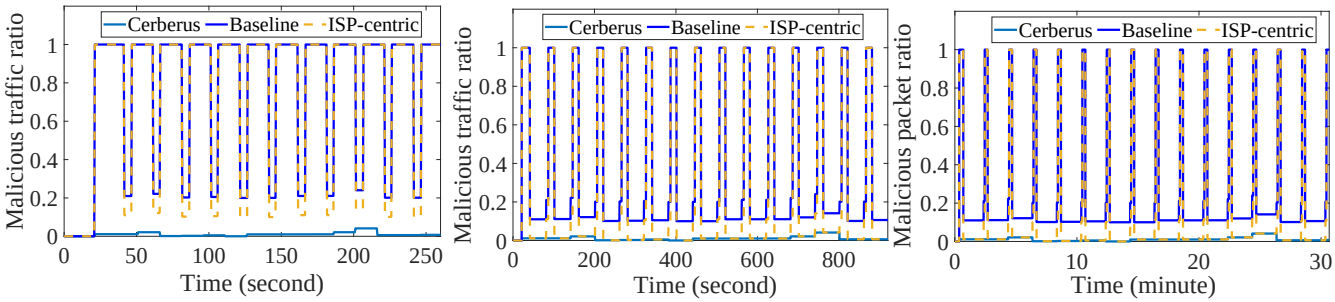


Figure 11: Resource usage of different programs (Cerb. contains 8 types of INM tasks), where the red line is the resource limit. For programs exceeding the red line, we have to manually split them and deploy them on multiple switches.

Takeaway: Cerberus can enhance the concurrency of the multi-vector DDoS defense system by 8 times. Cerberus

6. For some functions such as DDoS defenses, the actual available SRAM or ALUs are lower than 100%. This is because their monitoring modules are usually deployed in later stages.



(a) Hybrid attack (Period = 15s) **(b)** Hybrid attack (Period = 60s) **(c)** Hybrid attack (Period = 120s)

Figure 12: Dynamic attack evaluation. Single attack picks an attack each time, and Hybrid attack picks multiple attacks each time.

can run 8 types of defenses concurrently in a switch, while the baseline requires 8 times the resources (e.g., ALUs) to achieve the same level of concurrency.

6.3. Cerberus's Adaptability

In this section, we evaluate the effectiveness of Cerberus in defending against hybrid and dynamic attacks. We also demonstrate that Cerberus provides adaptability to other existing INM systems, such as NetWarden, which defends against covert channel attacks [28].

6.3.1. Hybrid and dynamic attacks. We have 16 types of candidate attacks (and 15 types of defenses), as shown in Table 1. We select 6 types of attacks for each period and vary their proportion from 10% to 40% of the available bandwidth (10 Gbps). We utilize real-world datasets [50] to replay background traffic at a speed of 1Gbps. To evaluate the fast adaptability, we choose three periods: 15 seconds, 60 seconds, and 120 seconds. Since some defenses (e.g., based on limiters) do not completely drop malicious packets, we use the remaining malicious traffic ratio to estimate the effectiveness of defenses.

We set a baseline that deploys at most 4 types of defense each time. Otherwise, the deployed defenses may not have enough space to achieve high accuracy. We also adopt an ISP-centric strategy in Jaqen [25], which leverages multiple switches to counter dynamic attacks. When the posture of attacks changes, Jaqen reconfigures a part of the switches at a time to meet the new requirements. To avoid introducing high false positives, all filters of activated switches are temporarily disabled during the reconfiguration process. In our settings, we use two programmable switches with multiple defense programs (Jaqen only uses one switch).

As shown in Figure 12, Cerberus can effectively counter hybrid and dynamic attacks and limit malicious traffic to a safe level. In contrast, the baseline cannot counter all types of attacks at the same time due to resource limitations. Therefore, the malicious traffic ratio is always high (0.2~0.3). The ISP-centric defense also faces challenges when the changing period is shorter than the reconfiguration time. This is because the attack types may have already changed before the ISP-centric defense completes its reconfiguration. As the changing period increases, the ISP-centric defense is able to follow the hybrid attack after a while, but the baseline still misses a portion of the malicious flows. Since the ISP-centric defense uses more switches than the baseline and Cerberus (Cerberus only uses one switch), this is reasonable. Unfortunately, both the baseline and the ISP-centric defense have a downtime window (with malicious traffic ratio = 1) during the reconfiguration process, as shown in Figures 12b and 12c. A tricky attacker may exploit it to disrupt the service of the defense systems. In contrast, Cerberus can reallocate memory within a few milliseconds without terminating the switches. When new attack types are detected, the control plane will allocate slices to corresponding functions and recycle idle slices, ensuring continuous and adaptive defense against attacks.

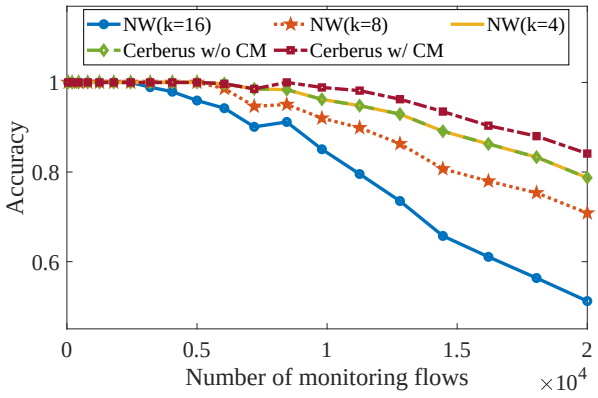


Figure 13: The accuracy of the estimation of IPD distributions.

6.3.2. Adaptable Netwarden. We develop a covert channel defender of Netwarden [28], which detects suspicious packets leaking secrets from a compromised host to outside networks. Due to existing firewalls, these packets may leverage covert channels such as covert timing channels. To detect covert timing channel attacks, the data plane records the distribution of the inter-packet delays (IPDs) of each flow. Then, the control plane can collect them for further analysis. To save memory, Netwarden uses k CMS counters

to record the distribution of IPD of connections. That is, $[0, t_1), [t_1, t_2), \cdots, [t_{k-1}, \infty)$.

Settings. In our experiment, we allocate a fixed memory space (2MB) to compare the accuracy of different kinds of strategies. We allocate two arrays with a width of 32-bit for each CMS counter (i.e., $d = 2, width = 32$). We set different k for the baseline (Netwarden), i.e., $k = 4, 8, 16$. For Cerberus, the k is adaptable and we set the initial value to 16. A larger k means a finer-grained distribution. However, a larger k also requires more CMS counters, which results in smaller counter sizes. We utilize real-world datasets [50] to replay background traffic at a speed of 1Gbps, which may intersect with the malicious flows and impact the final accuracy. Additionally, we generate a substantial volume of malicious flows that embed secrets using the timing channel.

As shown in Figure 13, most strategies perform well when the number of monitored flows is less than 3,000. However, as the number of monitored flows further increases, NetWarden (NW) with higher k (e.g., $k = 8, 16$) produces low accuracy. On the other hand, NW (k=4) produces high accuracy, but it only contains 4 CMS counters, which provide a coarse-grained view of the IPD distribution. As for Cerberus without applying the co-monitoring mechanism (Cerberus w/o CM), we do not change the length of states. We simply change the size of each register. For example, we set the size of each register to 2^{14} and k to 16 when there are a small amount of monitored flows. When the number of monitored flows increases, we increase the size to 2^{15} ($k = 8$) and 2^{16} ($k = 4$). Therefore, Cerberus w/o CM always outperforms a fixed NetWarden strategy. Moreover, we estimate the accuracy of Cerberus with the co-monitoring mechanism (i.e., Cerberus w/ CM). When we need to monitor more flows, the control plane will shorten the length of each state (e.g., from 32-bit to 8-bit). As a result, Cerberus does not need to reduce the number of counters (i.e., k) while still producing high accuracy. In our experiment, Cerberus always provides a fine-grained view ($k = 16$) of the IPD distributions of each flow.

Takeaway: Cerberus can enhance the adaptability of existing P4-based INM systems to cope with changing network conditions.

6.4. Cerberus's Overhead

In this part, we estimate the overhead of Cerberus, including the packet recirculation rate, the bandwidth utilization, and processing latency. The recirculation packet rate can affect the throughput of the programmable switch. Basically, if a packet is recirculated, it will create a new packet, and they totally go through the pipeline three times. That is, if the packet recirculation rate is 100%, the throughput will drop to 1/3. Besides, the bandwidth of the data-to-control channel is 10Gbps. As for the processing latency, it is mainly affected by the pipeline length. It is worth noting that Cerberus does not delay any packets, even if the packets are recirculated. The original packet is forwarded immediately, and the mirror packet is recirculated. Therefore, the recirculation does not increase client latency.

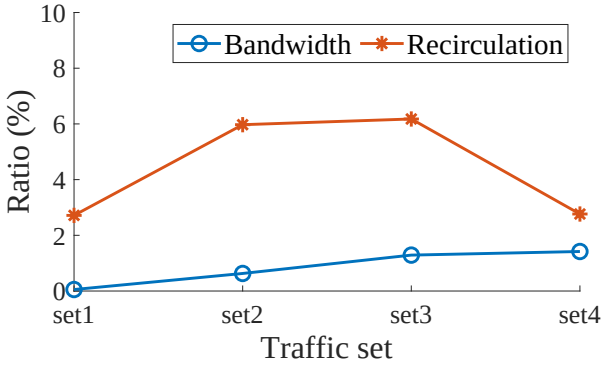


Figure 14: The Overhead of the co-monitoring mechanism on the host rate counter.

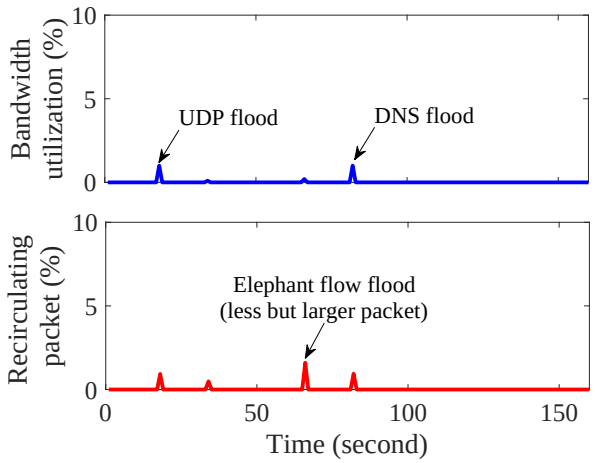


Figure 15: The Overhead of the co-monitoring mechanism on hybrid and dynamic attacks.

Host rate counter. We first evaluate the overhead of the host rate counter. To enlarge the available memory, the data plane uses the co-monitoring mechanism to share the workload with the control plane. When Cerberus runs the host rate counter, it dynamically adjusts the workload distribution. For example, when there are fewer hosts and lower throughput (e.g., traffic set 1), Cerberus allocates a long length (6 bits) for the host rate counter. When the number of hosts and the throughput increase, Cerberus also enlarge the size by reducing the length in the data plane (e.g., from 6 bits to 4 bits). Therefore, the consumed bandwidth increase as the size of the replayed traffic set increase, as shown in Figure 14. Interestingly, we obverse that the recirculation ratio decreases when we replay the largest traffic set (set 4). Since the replaying speed is higher, there are more passing packets. Meanwhile, the number of recirculated packets is always below a threshold due to the adjustment of Cerberus's resource manager.

Adaptable DDoS defenses. We further evaluate the overhead of hybrid and dynamic defenses. In our experiments, we choose six types of attacks, including ICMP flood, UDP flood, Slowloris attack, HTTP flood, elephant flow, and DNS

flood. All of them require defenders to count the number of related packets, so the length of these states can be very high. Therefore, it is a suitable scenario to show the overhead of the co-monitoring mechanism. In each period (e.g., 15s), we randomly choose one to three attacks from six types of attacks, and their proportions vary from 10% to 100%. We do not launch more attacks at a time because the volume of each attack can become too low to cause a negative consequence. When detecting occurring attacks, Cerberus will allocate memory for corresponding functions. By default, we set the length of states of 5 defenses to 8, except for the elephant flow defense, which has a length of 16.

Overall, the consumed bandwidth of the data-to-control channel is quite low, as shown in Figure 15. We notice that the UDP flood and DNS flood attacks cause the highest bandwidth overhead, i.e., 1% (0.1Gbps/10Gbps). This is because the default length of their states is too short to record the states of high-rate flows. Thus, many carry flags are sent to the control plane. However, the control plane immediately adjusts the length of slices, and the bandwidth utilization rate reduces quickly (within 1 second). Besides, the peak value of recirculating packets is 1.6% when attackers launch a single elephant flow flood attack. We further analyze the number of recirculating packets during this time window. We find that the recirculating packets are less than the recirculating packets during UDP flood and DNS flood attacks. This is because the elephant flow flood prefers to generate large packets to consume the bandwidth. At the same time, the attack does not necessarily generate too many malicious packets and can escape from a high pps filter (e.g., UDP flood filter). With a much lower total number of passing packets[7], its recirculating packet ratio becomes the highest. Finally, Cerberus reduces the overhead by recording the elephant flows on an elephant region with a full length (i.e., 32-bit). Since the number of elephant flows is relatively small, a region with a size of 8192 and a length of 32 bits is enough.

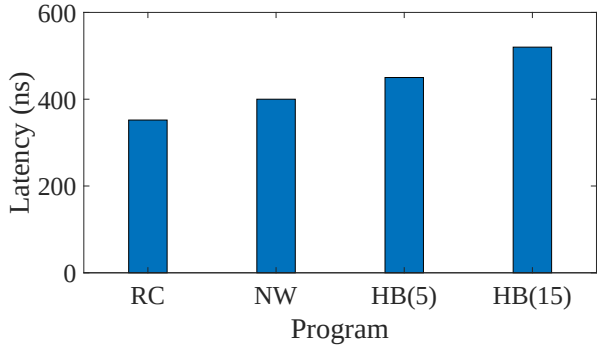


Figure 16: The processing latency of different programs.

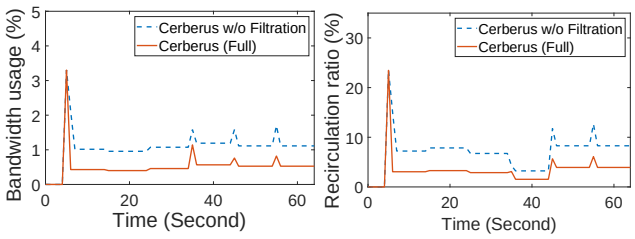
Latency. We evaluate the processing latency of Cerberus for different programs. Based on programmable switches, Cerberus can run INM functions with extremely low processing latency, as shown in Figure 16. Here, RC means a host rate counter, NW means NetWarden, HB (5) means a hybrid DDoS defense program containing 5 types of defense (SYN flood, ACK flood, ICMP flood, elephant flow, and UDP flood), and HB (15) contains 15 types of defenses.

7. In this case, most passing packets are benign, although they consume less bandwidth

6.5. Cerberus's Robustness

To demonstrate Cerberus's robustness, we conduct experiments where attackers launched various attacks to disrupt the functionalities of Cerberus. By deliberately causing overflows, packets are uploaded to the control plane and recirculated to switches. The main goal of these attacks is to increase the frequency of overflows. Consequently, the attackers are able to exhaust the bandwidth resources of the data-to-control plane channel and significantly reduce the throughput of the data plane.

Settings. Unlike previous DDoS attacks, we avoid sending low-rate flows that are ineffective in triggering overflow, as well as elephant flows due to their limited numbers and susceptibility to filtration by Cerberus's filtration region. Instead, we enable attackers to generate flows with moderate rates. Specifically, TCP flows are generated at 1Mbps/128pps per flow, UDP flows at 1Mbps/256pps per flow, ICMP flows at 512Kbps/1Kpps per flow, HTTP flows at 512Kbps/128pps per flow, and DNS packets at 512Kbps/128pps per flow. Moreover, we allow attackers to dynamically change the types and proportions of malicious flows, with a maximal throughput of 10Gbps for malicious flows. To represent realistic scenarios, we utilize real-world datasets for background traffic, replayed at 1Gbps. To minimize the impact on the elephant region, we ensure an even distribution of the flow rate. Additionally, the attacker changes the proportion of malicious traffic every 10 seconds. In our evaluation, we employ two solutions. The first one is **Cerberus without Filtration**, where we do not utilize the elephant region to filter top-K elephant flows. The second solution, **Cerberus (Full)**, incorporates an elephant region with a size of 8192 (approximately 256Kb memory) to effectively filter and handle top-K elephant flows.



(a) Bandwidth usage **(b)** Recirculation rate

Figure 17: Cerberus's overhead under attacks

As shown in Figure 17, the bandwidth usage and recirculation rate experience a significant surge (>3%, >20%) at the beginning of the attacks. However, thanks to the resource manager in Cerberus, the length of slices is adjusted to

mitigate overflow occurrences. Notably, we observe that the overhead reduces to a low level after one second. Compared to **Cerberus without Filtration**, **Cerberus (Full)** demonstrates a greater reduction in overhead. This improvement is attributed to the ability of **Cerberus (Full)** to capture a portion of malicious flows using the elephant region, effectively preventing them from triggering the overflow. We also observe that the overhead exhibits fluctuations every 10 seconds. As the proportion of packets changes, the original settings may deviate from the optimal configuration. Therefore, Cerberus requires a brief period of time (about 1s) to calculate and apply new settings. This explains why we observe constant fluctuations in the values of overhead metrics. Despite the fluctuations, the overhead consistently remains at a relatively safe level, with values below 1.5% for bandwidth usage and below 7% for the recirculation ratio.

7. Related work

Traditional network monitoring systems. Traditional network monitoring systems are either based on centralized collection information or rely on proprietary hardware to analyze passing flows locally. For the first type of network monitoring system [16]–[23], they usually leverage software-defined networking (SDN) technology. However, their performance is too poor to handle large-scale traffic sets. On the other hand, some network monitoring systems are based on proprietary hardware. For example, many switches support firewall rules and access control lists (ACLs). Network operators can define customized rules to analyze the fields or types of packets [53], [54]. However, these approaches may not be accurate when hosts or applications use non-standard values (e.g., do not use port 80 for HTTP). Moreover, some packets may contain new protocols or new packet headers that cannot be recognized by switches. Therefore, they are too inflexible to handle various and changing network traffic.
INM systems based on programmable ASIC. Emerging programmable switches are becoming desirable choices for large-volume stream processing due to their advantages of flexibility, performance, and cost-efficiency. There are a lot of industrial and academic efforts to implement programmable switches for better QoS [13], [42]–[45] and safety guarantee [25]–[27].

Unfortunately, the memory of current programmable switches is insufficient to handle large-scale INM tasks concurrently. Most systems hence design a series of mechanisms to reduce memory usage or provide adaptability. For example, when faced with new requirements, Poseidon [24] reroutes all network flows to servers with lower throughput during reconfiguration of the programmable switches. The performance degradation makes it unsuitable for high-performance INM systems. Jaqen [25] tries to build an ISP-centric programmable DDoS defense system. With more programmable switches, network managers can reconfigure switches for new requirements. To avoid introducing high false positive rates, Jaqen has to disable filters during the reconfiguration process, which might be exploited by an advanced attacker to trigger the reconfiguration of all switches. Omnimon [29] is a distributed system that assigns multiple switches to track flows cooperatively. However, Omnimon is not transparent to hosts because it requires end-hosts to participate in the flow-tracking process. Moreover, the efficiency of Omnimon depends on the number of switches on the path. If too many flows are on a short path, the accuracy can drop to a low level. Bedrock [55] provides a secure foundation for RDMA systems by using programmable switches. Bedrock designs CPU-bypassing defense primitives to analyze the passing flows and counter a series of RDMA attacks. Due to resource limitations, Bedrock runs different defenses separately instead of concurrently. BeauCoup [35] leverages the coupon collector that randomly updates one of the queried features. With a small constant of memory accesses, it can support multiple distinct counting queries simultaneously. However, BeauCoup introduces relatively high error rates. Therefore, BeauCoup is not suitable for INM tasks requiring high precision.

8. Discussion

Possible attacks. An adversary may try to increase the overhead of the co-monitoring mechanism. In Section 6.5, we show that the resource manager of Cerberus can adjust the lengths of the slices and the filtration rules to reduce overhead. On the other hand, attackers may fail to launch many DDoS attacks due to conflicting settings. For example, the Slowloris attack aims to exhaust the resource of connection pool resources by employing numerous unique low-rate flows. However, this conflicts with the requirement of generating medium-rate flows due to resource constraints.
Various operations. The memory slicing mechanism can merge different features into one register, but their operators need to be converted to the same one. Although some operators, such as addition and subtraction, can be converted into each other, we cannot merge all types of operations into a single operation. In the worst cases, all INM tasks require unique operations, then Cerberus cannot enhance the concurrency. Even so, Cerberus can still enhance the capacity and adaptability for INM tasks.
Recirculation and uplink bandwidth. The co-monitoring mechanism introduces recirculation, which can affect the throughput of the data plane and increases the communication overhead between the data and control planes. However, we argue that these effects are acceptable. First, the impact is short-term, because Cerberus's resource manager can mitigate it after a while (e.g., several seconds). Second, when there is heavy overhead due to the co-monitoring mechanism, it may indicate that the load of the current network is too high. In such cases, we can sacrifice a small part of the performance to maintain good functionality. Besides, we can actively reroute a part of the traffic to mitigate the impact.

9. Conclusion

In this paper, we design Cerberus, an efficient and effective INM system based on programmable switches. We

abstract INM tasks into K-F pairs and design a novel memory slicing mechanism to share resources between multiple K-F pairs. We further improve the capacity of Cerberus by proposing a co-monitoring mechanism. We design a novel resource manager to support resource reallocation for dynamic requirements. We implement a prototype of Cerberus and conduct comprehensive evaluations to demonstrate that Cerberus can handle INM tasks efficiently and effectively. Moreover, Cerberus can adapt to new or fast-changing requirements without interrupting programmable switches.

Acknowledgements

We want to thank the anonymous reviewers for their valuable comments. This material is based upon work supported in part by the National Science Foundation (NSF) under Grant No. 1700544, 2148374, and 2226339, DHS Grant No. 518700-00001, and ONR Grant No. N00014-20-1-2734. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF, DHS, and ONR.

References

[1] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[2] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch." in *NSDI*, vol. 13, 2013, pp. 29–42.

[3] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings 14*. Springer, 2011, pp. 161–180.

[4] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, 2013, pp. 25–30.

[5] J. R. Ballard, I. Rae, and A. Akella, "Extensible and scalable network monitoring using opensafe." *Inm/wren*, vol. 10, 2010.

[6] (2022) Internet traffic volume. [Online]. Available: https://www.ibisworld.com/us/bed/internet-traffic-volume/88089/

[7] P. Nicholson. (2018) 5 most famous ddos attacks. [Online]. Available: https://www.a10networks.com/resources/articles/5-most-famous-ddos-attacks

[8] S. Moss. (2016) Major ddos attack on dyn disrupts aws, twitter, spotify and more. [Online]. Available: https://www.datacenterdynamics.com/en/news/major-ddos-attack-on-dyn-disrupts-aws-twitter-spotify-and-more/

[9] A. Scroxton. (2016) Dyn reveals details of complex and sophisticated iot botnet attack. [Online]. Available: http://book.itep.ru/depository/ddos/Dyn_reveals_details_of_complex_and_sophisticated_IoT_botnet_attack.htm

[10] D. Pauli. (2016) Chinese gambling site served near record-breaking complex ddos. [Online]. Available: https://www.theregister.com/2016/07/01/470_gbps_multivector_chinese_gambling/

[11] C. Security. (2018) Ddos attacks 2018: New records and trends. [Online]. Available: https://www.calyptix.com/research/ddos-attacks-2018-new-records-and-trends/

[12] W. Turton. (2014) An interview with lizard squad, the hackers who took down xbox live. [Online]. Available: https://www.dailydot.com/debug/lizard-squad-hackers/

[13] 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 *SIGCOMM '17*.

[14] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using openflow," *ACM Sigcomm Demo*, vol. 4, no. 5, p. 6, 2009.

[15] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–13.

[16] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *USENIX Security '15)*.

[17] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Yau, and J. Wu, "Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis," in *TIFS '18*.

[18] M. Zhang, J. Bi, J. Bai, Z. Dong, Y. Li, and Z. Li, "Ftguard: A priority-aware strategy against the flow table overflow attack in sdn," in *Proceedings of the SIGCOMM Posters and Demos*, 2017, pp. 141–143.

[19] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[20] T. Xu, D. Gao, P. Dong, C. H. Foh, and H. Zhang, "Mitigating the table-overflow attack in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1086–1097, 2017.

[21] M. Zhang, J. Bi, J. Bai, and G. Li, "Floodshield: Securing the sdn infrastructure against denial-of-service attacks," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 687–698.

[22] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 239–250.

[23] Y. Xu and Y. Liu, "Ddos attack detection under sdn context," in *IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications*. IEEE, 2016, pp. 1–9.

[24] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *NDSS '20*.

[25] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *USENIX Security '21*.

[26] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *USENIX Security '21*.

[27] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi *et al.*, "Flow event telemetry on programmable data plane," in *SIGCOMM '20*.

[28] J. Xing, Q. Kang, and A. Chen, "Netwarden: Mitigating network covert channels while preserving performance," in *USENIX Security*, 2020.

[29] Q. Huang, H. Sun, P. P. Lee, W. Bai, F. Zhu, and Y. Bao, "Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy," in *SIGCOMM '20*.

[30] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with sdn data plane," in *IEEE INFOCOM 2017-IEEE conference on computer communications*. IEEE, 2017, pp. 1–9.

[31] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan, "Nethcf: Enabling line-rate and adaptive spoofed ip traffic filtering," in *2019 IEEE 27th international conference on network protocols (ICNP)*. IEEE, 2019, pp. 1–12.

[32] J. Bai, J. Bi, M. Zhang, and G. Li, "Filtering spoofed ip traffic using switching asics," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018, pp. 51–53.

[33] G. Grigoryan and Y. Liu, "Lamp: Prompt layer 7 attack mitigation with programmable data planes," in *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, 2018, pp. 158–159.

[34] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 19–27.

[35] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "Beaucoup: Answering many network traffic queries, one memory update at a time," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 226–239.

[36] (2023) Cerberus. [Online]. Available: https://github.com/successlab/Cerberus

[37] B. Claise, "Cisco systems netflow services export version 9," Tech. Rep., 2004.

[38] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 245–256, 2004.

[39] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Application of sampling methodologies to network traffic characterization," in *Conference proceedings on Communications architectures, protocols and applications*, 1993, pp. 194–203.

[40] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 2004, pp. 628–635.

[41] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors."

[42] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *NSDI '18*.

[43] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *NSDI '20*.

[44] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *SOSP '17*.

[45] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "Netchain: Scale-free sub-RTT coordination," in *NSDI '18*, 2018, pp. 35–49.

[46] M. Zhang, G. Li, X. Kong, C. Liu, M. Xu, G. Gu, and J. Wu, "Nethcf: Filtering spoofed ip traffic with programmable switches," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[47] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[48] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[49] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[50] W. Project. Mawi working group traffic archive. [Online]. Available: http://mawi.wide.ad.jp/mawi/

[51] "Barefoot® Tofino™ ". https://www.barefootnetworks.com/technology/#tofino.

[52] (2022) 2018. barefoot p4 studio. [Online]. Available: https://www.barefootnetworks.com/products/brief-p4-studio/

[53] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 648–656.

[54] W. Jiang and V. K. Prasanna, "A fpga-based parallel architecture for scalable high-speed packet classification," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2009, pp. 24–31.

[55] J. Xing, K.-F. Hsu, Y. Qiu, Z. Yang, H. Liu, and A. Chen, "Bedrock: Programmable network support for secure {RDMA} systems," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2585–2600.

Appendix A. Meta-Review

A.1. Summary

This paper proposes an in-network security monitoring (INM) system for programmable networks that mitigates challenges with high-volume DDoS attacks. The authors use memory slicing and resource management mechanisms to dynamically handle high-volume traffic, and they implement and evaluate their solution using P4 switches.

A.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

A.3. Reasons for Acceptance

1) This paper creates a new tool, Cerberus, that enables the implementation of multiple in-networking monitoring tasks over high-volume traffic with programmable data planes.
2) This paper shows how the control and data plane can collaboratively and dynamically support INM tasks that may require different resources based on the network traffic in question. Prior approaches have only focused on improving certain INM tasks through sketching.
3) The proposed tool provides a valuable step forward. Cerberus allows for an efficient and adaptable INM solution, with no down time, which is a clear step forward in the domain of INM.

A.4. Noteworthy Concerns

The authors position the proposed solution as a general approach towards optimizing memory constraints in programmable networks with dynamic scalability, which generalizes to a broader problem in networking problem rather than specifically a security problem. The proposed approach does, however, fall within scope of security because performance issues can very quickly become security (availability) issues.