CHAMALEONET: Programmable Passive Probe for Enhanced Visibility on Erroneous Traffic

Zhihao Wang, Alessandro Cornacchia, Andrea Bianco, *Senior Member, IEEE*, Idilio Drago, Paolo Giaccone, *Senior Member, IEEE*, Dingde Jiang, *Member, IEEE*, and Marco Mellia, *Fellow, IEEE*

***Abstract*—Traffic visibility remains a key component for management and security operations. Observing erroneous traffic, i.e., *unanswered requests or error messages*, is fundamental to detecting misconfiguration, temporary failures or attacks. CHAMALEONET transforms any production network into a transparent monitor to let administrators collect such erroneous traffic. CHAMALEONET is programmed to ignore well-formed traffic and record only erroneous packets, including those generated by misconfigured or infected internal hosts, and those sent by external actors which scan for services. Engineering such a system poses several challenges, from scalability to privacy. Leveraging the Software-Defined Networking (SDN) paradigm, CHAMALEONET processes the humongous amount of traffic flowing through the network border and focuses on erroneous packets only, lowering the pressure on the collection system. Moreover, it offers traffic anonymisation to conform to privacy regulations. CHAMALEONET enables the seamless integration with active deceptive systems like honeypots that can impersonate hosts/ports/services and engage with senders. In an operational scenario, we show that the SDN in-hardware filtering reduces the traffic to the controller by 90% , resulting in a scalable solution, which we offer as open source. Simple statistical analytics unveil the precious information carried by erroneous traffic. We discover internal misconfigured and infected hosts, identify temporary failures, and show enhanced visibility on attackers' scanning activities that look for vulnerable services.**

***Index Terms*—Traffic visibility, network monitoring, software-defined networking, programmable data plane.**

I. INTRODUCTION

Network monitoring has always been the first step to implement network management or security policy. Some monitoring systems offer visibility on regular traffic to derive statistics on performance and application usage, e.g., flow loggers [1], [2], [3]; Some systems focus on protecting the network infrastructure, e.g., intrusion detection/prevention systems (IDS/IPS or IDPS) [4], [5]; Some collect unsolicited traffic, e.g., network telescopes [6], [7].

Yet live networks contain well-formed flows, i.e., requests that get answered by the destination. And flows that remain unanswered or generate error messages at the network layer. We term the latter *erroneous traffic*. Such traffic arises when clients attempt to connect to offline (or firewalled) systems, or try to reach unintended or unavailable servers (possibly caused by routing anomalies). At last, malicious actors scanning networks and services generate erroneous traffic.

Prior work [8], [9], [10] has shown erroneous traffic reveals interesting insights. For example, Richter et al. [9] show that unsolicited traffic in large CDNs is not uniformly random: over 30% of scans are localized, targeting few address regions. Similarly, DScope [8] reports up to $450\times$ more scan traffic than expected under random scanning. These results indicate that erroneous traffic in production networks reflects systematic targeting and structural bias rather than background noise. This motivates detecting and collecting such traffic in production environments to study targeting effects and distribution patterns. Our previous work [11] also reveals the value of outbound erroneous traffic in identifying signals of misconfiguration, transient failures, or attacks.

Similarly, traditional network telescopes record all packets destined to an *unused subnet*, where no host is connected. Any packet they receive is erroneous by definition, and is limited to the so-called Internet radiation [6]. Telescopes operate on reserved and unassigned address spaces, devoting precious IPv4 subnets for pure data collection. Here, we transform any production network into a better-telescope as we avoid reserving IP addresses for monitoring purposes, and collect erroneous traffic generated by external and internal hosts.

At last, traditional flow loggers operate on live networks but provide only statistical summaries of flows, without distinguishing erroneous traffic. IDPS inspect packets against predefined rules or known signatures. Although not designed to collect erroneous traffic, IDS/IPS can, in principle, be extended to track it by maintaining per-flow states (e.g., triggering events for new requests and timeouts [12]), since they inherently support stateful processing. However, stateful operation is costly, and their support for raw packet logging is limited: the scale of typical data rates challenges the usage of IDS/IPS to log erroneous traffic [1], [3], [13].

Zhihao Wang is with the School of Aeronautics and Astronautics, University of Electronic Science and Technology of China, Chengdu 611731, China, and the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy (e-mail: zhihao.wang@polito.it).

Alessandro Cornacchia is with Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, 23955 Thuwal, Saudi Arabia (e-mail: alessandro.cornacchia@kaust.edu.sa).

Andrea Bianco and Paolo Giaccone are with Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Turin, Italy (e-mail: andrea.bianco@polito.it; paolo.giaccone@polito.it).

Idilio Drago is with the Computer Science Department, Università di Torino, 10124 Turin, Italy (e-mail: idilio.drago@unito.it).

Dingde Jiang is with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Laboratory of Electromagnetic Space Cognition and Intelligent Control, Beijing 100091, China, and also with the Digital Media Art, Key Laboratory of Sichuan Province, Sichuan Conservatory of Music, Chengdu 610041, China, and also with the Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China (e-mail: jiangdd@uestc.edu.cn).

Marco Mellia is with the Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy (e-mail: marco.mellia@polito.it).

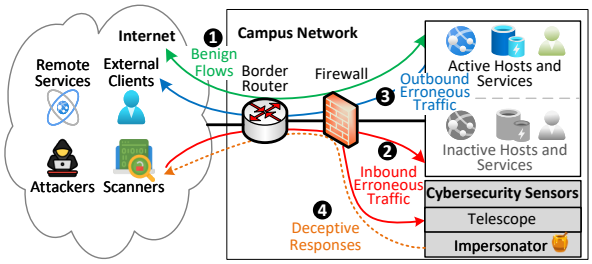


Fig. 1: Erroneous traffic to/from a campus network is steered to cybersecurity collectors (e.g., telescope or honeypots).

Collecting erroneous traffic in production networks is inherently challenging. Erroneous flows are defined by the absence of a response within a detection window, a condition that cannot be identified through static rules or signature-based matching used by traditional monitoring systems. Detection therefore requires retrospective, stateful tracking of bidirectional flows and temporary buffering of initial packets. At the same time, most flows are well-formed; without early suppression, redundant packet processing would unnecessarily consume resources in the monitoring pipeline. Furthermore, host and service liveness continuously evolves, and the system must promptly capture traffic destined to hosts or services that have just gone offline, which requires dynamic rule installation and expiration.

These constraints call for a separation between high-speed packet filtering and scalable timeout-based stateful logic. The SDN architecture naturally enables this design: programmable switches perform line-rate suppression of benign traffic, while the controller executes timeout-based tracking and manages rule lifecycle operations. At the same time, network programmability allows flexible steering of erroneous traffic (e.g., toward honeypots) and supports privacy-aware data minimization, addressing operational and regulatory requirements.

For this, we propose CHAMALEONET, a flexible system that opportunistically allows the collection of erroneous traffic in an operational network. Fig. 1 depicts CHAMALEONET goals: it ignores all regular traffic (green arrow ❶) and collects only those packets for which no valid response is observed (red arrows ❷ ❸). CHAMALEONET logs the erroneous traffic generated by *external hosts* (inbound erroneous traffic ❷) by opportunistically logging unanswered requests or error messages destined to addresses assigned to unreachable internal hosts. Likewise, CHAMALEONET logs requests originated by *internal hosts* (outbound erroneous traffic ❸), unveiling issues with external services and offering visibility on misconfigured or infected internal hosts, thereby facilitating troubleshooting and repair. Furthermore, CHAMALEONET can transform itself into an active impersonator (orange dashed arrow ❹), impersonating inactive hosts or services, as a honeypot system.

Engineering CHAMALEONET requires ingenuity. We leverage Software Defined Networks (SDN) and programmable switches to transform a live network into a flexible monitor. When services or hosts are not responding, we forward their traffic to back-end systems, being it a simple passive collector or an active impersonator that can impersonate the destination. Conversely, when observing a regular flow, the programmable switch filters all subsequent packets to reduce the collector load and preserve live traffic privacy. To detect at runtime services that are not respondents, we introduce a dedicated stateful *Flow-State Detection Network Function* (FSD-NF). In summary, CHAMALEONET transforms any network into a flexible and transparent monitor for erroneous traffic, enabling visibility on a wide range of management and cybersecurity events coming from external and internal hosts. We offer CHAMALEONET to the community as open source.[1]

We design CHAMALEONET following the data minimisation principle to respect privacy and ethical principles. We collect only erroneous packets, suspicious by definition, and avoid logging well-formed traffic. By offloading the filtering and anonymisation features to the programmable switch, CHAMALEONET enables monitoring tens of Gbps with a single off-the-shelf server. Traditional sharding and flat controller solutions can be easily adopted [14] to scale the system.

We deployed CHAMALEONET at our University campus network for months and next we will present some findings. In summary, we contribute the following:

- We introduce the concept of erroneous traffic and design CHAMALEONET, a system that transforms any private network into a flexible monitor of erroneous traffic.
- We implement CHAMALEONET atop SDN principles to transparently filter, anonymise, and steer traffic, respecting users' privacy. Offloading the filtering of regular traffic to the switch reduces the controller load by 90% .
- CHAMALEONET improves visibility on external Internet radiation while empowering the administrators to engage with scanners by selectively enabling impersonators.
- Simple statistical analysis of the erroneous traffic patterns expose suspicious internal hosts, misconfigured systems, and routing issues, which would otherwise go unnoticed.

Next, Sec. II presents the related work. Sec. III presents the design and architecture of CHAMALEONET. Sec. IV describes its implementation. Sec. V reports on its deployment and key findings, while Sec. VI evaluates its performance in a controlled environment. Sec. VII outlines strategies to improve the architecture. We discuss the ethical considerations in Sec. VIII and conclude the paper in Sec. IX.

II. RELATED WORK

A. Passive measurement mechanisms

Several mechanisms exist to passively monitor traffic, from simple network telescopes to IDS/IPS or IDPS. Telescopes collect packets sent to subnets where neither servers nor clients are connected. The collected Internet background radiation [6], [7] gives visibility on Internet scanners, botnets, coordinated attacks, failures, routing issues and even censorship policies [15], [16]. Telescopes ignore regular and some erroneous traffic and waste resources due to the need to leave unused large ranges of IP addresses [17].

Recent works have explored flexible telescope designs. In [9] authors leveraged CDN infrastructure to study unsolicited traffic reaching production servers, collecting the unsolicited packets blocked at the firewalls of CDN servers.

[1] https://github.com/zhihao1998/ChamaleoNet

TABLE I: Scope of CHAMALEONET

	Traffic	Focus	Pkt	Flow	Alert	Reply	Rules
Telescope	In	Dark	✓				static
Reactive Telescope	In	Dark	✓			✓	static
Flow monitor	In/Out	All		✓			static
IDPS	In/Out	Malicious	✓	✓	✓		static
CHAMALEONET	In/Out	Erroneous	✓			✓	dynamic

Attracted by live CDN nodes, attackers send a variety of packets that are not observed in telescopes. Similarly, DScope [8] places telescopes in cloud data centers. It continually and dynamically allocates new IP addresses from the Amazon Web Services, retaining them for 10 minutes to collect inbound TCP connection attempts. Meta-telescope [10] infers unused Internet address blocks and captures the corresponding traffic without requiring administrative control over those blocks.

These designs require that addresses remain *unassigned/unused* when used for collecting unsolicited traffic. As such, they lack visibility into: *i)* addresses that are temporarily offline but not yet incorporated into the pool, and *ii)* addresses with only a subset of ports active. In contrast, CHAMALEONET immediately gains visibility of a host once it goes dark and continuously monitors the unused ports of *active* hosts.

Reactive telescopes such as Spoki [18] and others [19] augment telescopes with the ability to respond to incoming requests through simple impersonators, e.g., opening the TCP connection to capture the first payload.

On the other extreme, IDPSes analyse live traffic in real time and block malicious or suspicious activities, eventually producing alerts for administrators [4], [5]. Snort, Suricata and Zeek are popular open-source solutions [20]. IDPSes react to attacks and system abuses, using known signatures or anomaly detection. They offer the ability to collect packet traces, selectively logging only packets causing the incident [21] or logging all packets.

In between, network flow loggers offer visibility on operational traffic [1], [2], [3], allowing application classification, performance monitoring, and network management in general. They process packets in real time and generate logs at the flow level. They offer very limited packet capture capabilities, typically meant to capture some samples of traffic or some specific protocols.

Table I summarises the CHAMALEONET scope. All platforms process incoming traffic. Flow monitors, IDPSes and CHAMALEONET offer visibility on traffic initiated by internal hosts (outgoing traffic). Telescopes focus on dark traffic, flow monitors summarize all traffic, while IDPSes are usually set to look for malicious traffic. Telescopes capture packets, while flow monitors log information at the flow level. IDPSes produce alerts based on rules, and have some packet capture abilities. Only reactive telescopes and CHAMALEONET support backend impersonators that can actively respond to incoming requests, such as completing the TCP three-way handshake.

B. Monitoring and defence mechanisms in SDN

The authors of [22] provide a comprehensive taxonomy of past work that leveraged the use of SDN as a defence mechanism. Closest to our work are those related to honeypot systems. The study in [23] introduced a honeynet based on SDN, akin to CHAMALEONET, which duplicates traffic to the honey-network for further examination. In [24], authors proposed a solution where the controller inspects suspicious traffic packets and redirects it to honeypots. The work in [25] introduced a DDoS mitigation system that leverages programmable data planes to adapt to dynamic attacks. It provides a tailored abstraction to express DDoS defence policies, shielding the underlying hardware complexities from programmers. None of this works can dynamically filter well-formed traffic at runtime and redirect only *erroneous* traffic to collector systems.

C. Stateful traffic processing with programmable switches

CHAMALEONET enables stateful processing of incoming traffic by leveraging the SDN switch and virtualized network functions. Some past works have focused on implementing a stateful processing only in the SDN switch, enabled by P4 [26] or OpenState [27]. The work in [28] proposed a reactive traffic control application that redirect traffic in real-time to a traffic classification engine. Differently from CHAMALEONET, the stateful logic is implemented internally to the SDN switch. The work in [29] considers a cybersecurity application leveraging a stateful approach directly in data plane. The work focuses only on the detection of TCP SYN flood attacks. The work in [30] proposes Detection as a Service (DaaS) architecture, in which the traffic is mirrored to a cluster of IDS engines. The SDN controller interacts with the IDS engines and with the switch to eventually block malicious flows. Differently from our solution, while the blocking decision is being determined, pending packets are not buffered, particularly the first packet, which is often crucial for identifying malicious behavior. Furthermore, the approach does not consider neither erroneous traffic nor traffic generated from internal sources.

Summary. CHAMALEONET is the only system that targets erroneous traffic. It places itself between passive telescopes and advanced flow monitors and IDPSes, offering visibility on both external and internal erroneous traffic. CHAMALEONET differs from previous works in several key aspects: 1) focuses on erroneous traffic, which previous systems largely ignore; 2) leverages SDN programmability to detect erroneous traffic through dynamic rules; 3) provides visibility into both incoming and outgoing traffic; 4) operates on live networks and transparently treats unused addresses – including temporarily offline hosts – as dark space; 5) extends the scope of dark space to the service level, i.e., unused ports on otherwise active servers; 6) logs traffic at the packet level while preserving privacy; 7) can selectively steer traffic to impersonators which can impersonate inactive hosts or specific services.

III. SYSTEM DESIGN

We show an overview of CHAMALEONET in Fig. 2. We design it as a pluggable network system (blue box) with minimal configuration. Connected to the campus network border router(s), CHAMALEONET observes all traffic entering/leaving the network and forwards only the erroneous portion to the

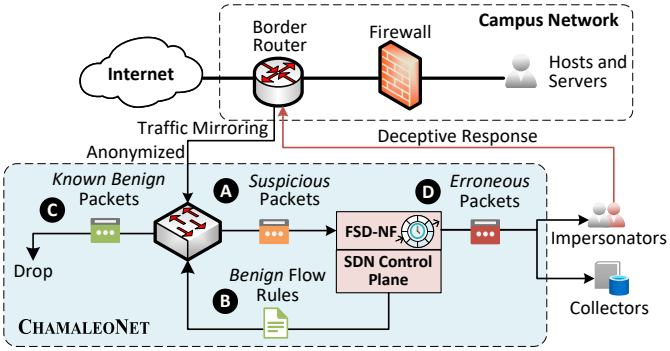


Fig. 2: Overview of ChamaleoNet architecture.

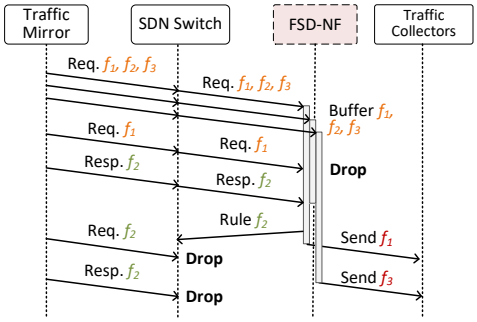


Fig. 3: ChamaleoNet detecting two flows with erroneous packets (f_1, f_3) and one benign flow (f_2).

collectors, eventually letting collectors impersonate the destination host or service. Observed the first packet of a new flow[2], it must determine whether the destination service is active, and eventually forward traffic destined to inactive services or hosts to the backend. ChamaleoNet must not cause any disruption or interference to the active services, nor forward production traffic payload to the collectors. Hence, we design the system to be *transparent* to services and benign traffic.

A. ChamaleoNet *Architecture*

ChamaleoNet leverages the SDN paradigm to separate the data plane, responsible for line-rate packet filtering, from the control plane, which manages the stateful logic to distinguish erroneous from benign traffic.

We here assume an out-of-path deployment (see Sec. VII for a discussion of the in-path alternative). All ingress and egress traffic, destined to and coming from the campus network, is mirrored to an SDN switch via either span ports or physical layer splitters. In case of multiple upstream providers, we assume all traffic is forwarded to handle asymmetric routing cases.

The switch implements a static per-service filter and a dynamic per-flow filter. When a packet arrives and does not match any filter, the switch forwards it to the SDN controller. Such a packet may either belong to a new legitimate flow or be potentially erroneous. Since both cases are plausible a priori, initially we denote these packets as *suspicious packets* (event Ⓐ in Fig. 2).

ChamaleoNet is designed to minimize logging of traffic destined to active services. To achieve this, a dedicated *Flow State Detection Network Function (FSD-NF)* determines whether a suspicious packet is *answered* (the service is active) or *unanswered* (the service is temporarily inactive). FSD-NF buffers the suspicious packet for a pre-defined DeTection (DT) timeout during which it waits for an eventual response packet that matches the same flow. Two cases can occur:

1) A response packet is received within the DT timeout: the service is deemed *active* and the suspicious packet (and following packets) are part of a legitimate flow. The controller drops the packet and installs a flow rule on the SDN switch (Ⓑ) to drop all matching packets. The SDN switch will discard any subsequent packet of the benign flow, avoiding overflowing the controller (Ⓒ).
2) The DT timeout expires: the service is deemed *inactive* and the suspicious packet is classified as *erroneous*. The controller forwards the original packet buffered at the FSD-NF to the cybersecurity collectors (Ⓓ), which may log or eventually reply to such packets acting as a honeypot.

Albeit easy to implement, we intentionally do not consider ICMP errors, like port, host or network unreachable messages, as a valid answer to form a flow. ChamaleoNet will thus consider both the first packet and the ICMP message as erroneous packets and send them to the collectors.

By observing all initial flow packets and responses, ChamaleoNet can keep track of which internal hosts are *alive*: Whenever the sender IP address corresponds to an internal host, ChamaleoNet marks the sending host as alive. This allows it to distinguish cases where external requests go unanswered because the destination is not alive (or dark, as in the case for telescopes) or present but refusing to answer (e.g., firewalled service or rebooting hosts). ChamaleoNet can take decisions on whether to respond to a packet, log or ignore it accordingly.

B. *Running Example*

The time diagram in Fig. 3 shows an example of how ChamaleoNet works. The switch receives traffic from the traffic mirror. It receives three packets belonging to three different flows (f_1, f_2, f_3). No rule matches these packets, so the switch forwards them to the FSD-NF, which buffers them and sets a DT timer to wait for more packets of the same flow to eventually arrive. Next, the FSD-NF observes a second request packet belonging to f_1 (e.g., a retransmission from the same sender), and drops it. Eventually, this second packet could be stored at the FSD-NF and sent to the cybersecurity collector too. Then, a response packet for f_2 arrives and is forwarded to FSD-NF, which deems the service and the host as active. FSD-NF marks the flow f_2 as "benign" and installs a rule in the switch to drop all subsequent packets belonging to f_2. In the meantime, FSD-NF drops all f_2 packets that may still be forwarded by the switch. At last, f_1 and f_3 DTs expire. The FDN-NF sets the corresponding buffered packets as "erroneous" and sends them to the collectors, removing any internal state related to them.

[2]We define a flow via the usual 5-tuple, up to the transport layer. Flows are bi-directional: packets matching the same 5-tuple belong to the same flow, regardless of their direction.

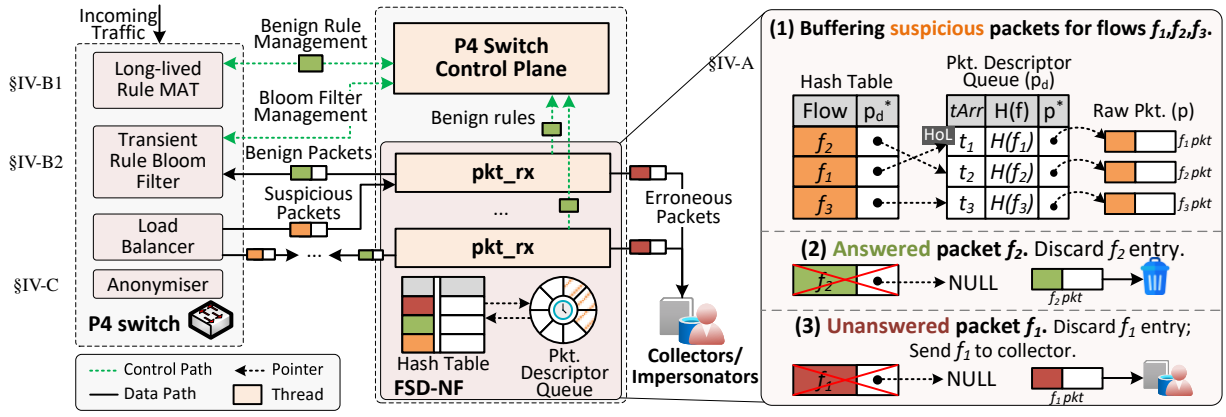


Fig. 4: Implementation of CHAMALEONET and its user-space packet processing workflow.

C. Assumptions and Threat Model

Assumptions. We make the following assumptions: 1) *Unified visibility:* the monitored network has unified borders and all ingress/egress traffic is mirrored to CHAMALEONET (out-of-path); 2) *Asymmetric routing:* in multi-homing settings, traffic is mirrored/forwarded such that asymmetric return paths remain visible. Note that asymmetric routing is an open challenge for such monitoring systems, e.g., in Meta-Tele-Scope [10]; 3) *Timing consistency:* packet timestamps and ordering at the monitoring vantage point are sufficiently accurate for enforcing the observation window; 4) *No inbound spoofing:* border devices are trusted and enables common filtering that prevents external hosts from spoofing internal source IPs (e.g., BCP38 [31]). 5) *Dynamic SDN-like rule support in the data plane:* the switch component—whether a physical SDN/P4 switch, a SmartNIC, or an eBPF/XDP program—accepts dynamic flow rules and forwards unmatched packets to the controller.

Threat Model. CHAMALEONET captures interactions in which requests reach routable addresses but fail to produce an observable reverse-direction response that can be associated with the original request within a defined observation window. Such interactions arise when endpoints are inactive, filtered, unreachable at the service layer, misconfigured, or non-responsive.

The threat model therefore consists of network activities that interact with such non-responsive endpoints, including scanning, probing, exploitation attempts, or unintended communications. These activities may be malicious or benign in intent. The defining property is the absence of an observable response within the observation window, not the intent of the sender. The goal of CHAMALEONET is to identify and analyse such interactions without disrupting normal traffic forwarding.

The model is limited to L3/L4 response observability within a configurable observation window. Interactions that successfully establish bidirectional transport semantics within the observation window are outside the erroneous traffic threat space, regardless of whether subsequent application-layer behaviors are anomalous, incomplete, or slow. An attacker could evade detection by injecting crafted reverse-direction packets that traverse the monitored border within the DT timeout. This requires coordinated control over the reverse path (e.g., control of both endpoints or reverse-path injection capability). In such cases, the interaction becomes indistinguishable from legitimate bidirectional L3/L4 communication and thus falls outside the erroneous-traffic model.

IV. IMPLEMENTATION

This section details the implementation of CHAMALEONET, whose architecture was introduced in Sec. III-A. Fig. 4 depicts the main components of CHAMALEONET and their interactions. We first describe the FSD-NF packet processing logic and data structures (Sec. IV-A), then present a two-tier rule management strategy for filtering benign flows at the SDN switch (Sec. IV-B), and finally highlight the traffic anonymizer (Sec. IV-C).

A. Flow State Detection Network Function (FSD-NF)

We rely on state-of-the-art per-flow monitoring based on multi-threading to implement the FSD-NF, which has been shown to cope with several tens of Gbps on off-the-shelf hardware [1]. For packet reception, CHAMALEONET uses any library that allows to capture packets. In our prototype, we use `libpcap` configured with "immediate mode", which disables packet batching and delivers packets to the FSD-NF upon arrival. Any other kernel-bypass packet I/O frameworks, such as DPDK [32], are transparent to CHAMALEONET design, and orthogonal to the scope of this work. At the FSD-NF, multiple pkt_rx threads process suspicious packets arriving from the switch, buffering packets and executing the FSD-NF packet processing logic described next. Suspicious packets are load-balanced across pkt_rx threads via hash-based selection on internal service identifiers, preserving per-service affinity; the load balancing hashing operation is offloaded to the switch.

1) Packet buffering and data structures: Fig. 4 shows the CHAMALEONET data structures and processing. There are three main data structures in FSD-NF: a hash table to store flow states; a packet descriptor queue to manage timeouts; a packet buffer to cache raw packets. The FSD-NF buffers suspicious packets and waits for a response (benign flow) or for the DT timer to expire (erroneous packet). We store packets

in FSD-NF memory and track their locations using descriptors organized as a ring buffer queue. The descriptor stores also the packet arrival time *tArr* and a pointer to the entry in the flow hash table. Event (1) in Fig. 4 (right) illustrates the relationship between the data structures for packets of flow f_1, f_2 and f_3, while Algorithm. 1 shows the pseudocode. When a new packet arrives and does not match any entry in the hash table, we store it in memory, create a new packet descriptor, set the $tArr$ field to the current time, and insert the packet descriptor it in the packet descriptor queue. To optimise the lookup when matching responses, we also store in the hash table a pointer to the packet descriptor position within the ring buffer.

2) Erroneous packet detection and DT timer management: CHAMALEONET detects erroneous packets by waiting for the DT timeout to expire. To avoid the overhead of managing multiple timers, one per packet, we implement a lazy timeout mechanism with a single timer. We exploit the simple observation that DT timeouts for suspicious packets expire in the same order in which the packets are received, and manage expiration via the following polling mechanism embedded within the pkt_rx thread (ln.20). The pkt_rx thread periodically (every P_{DT}) scans the packet descriptor queue starting from the entry with oldest $tArr$. For each entry, pkt_rx compares the current wall-clock time with *tArr*. If the elapsed time is greater than the DT timeout, it deletes the hash table entry, sends the packet to the collector and frees the corresponding packet descriptor. We stop at the first descriptor that has not exceeded the DT timeout. Event (3) in Fig. 4 (right) illustrates a packet of flow f_1 marked erroneous and sent to the collector. To limit the computation cost of this linear search operation, we allow a maximum search depth equal to $d_{\max}$ positions.[3]

3) Benign flow detection: if FSD-NF receives a response packet before the DT expiration, the flow is deemed as *benign*—i.e., internal service is active (Sec. III-A). FSD-NF deletes the corresponding suspicious packet from the packet descriptor queue, and installs an entry in the switch to filter all future packets of this benign flow. Response packets are detected in constant time using hash table lookups, by matching the response packet's flow identifier to the corresponding suspicious packet's flow entry.

B. Filtering Rules Management

The FSD-NF dynamically installs filtering rules in the switch so that subsequent packets belonging to benign flows are dropped. Filtering rules take the form of exact-match rules in MATs or approximate matching in Bloom filters. We refer to the mechanism as a dual-channel rule management. A *fast-path* channel quickly pushes filtering rules to the switch using in-band packets that update Bloom filters via the switch dataplane. A *slow-path* channel manages the installation of long-lived rules for well-known flows via the control plane. The terminology "fast" and "slow" refers to the latency of rule installation in the switch, which is negligible for Bloom filters and can be up to hundreds of milliseconds for MATs.

[3]While this check could be done in a separate thread, it would create race conditions on the ring buffer, introducing synchronisation overheads. Experimental results confirm that it is not needed (see Sec. VI).

Algorithm 1 pkt_rx and DT Timer Management in FSD-NF

Require: Packet descriptor queue Q, hash table H, DT timeout, polling period P_{DT}, max search depth $d_{\max}$
Ensure: Suspicious packets p belonging to flow f forwarded to collector, expired descriptors freed

On new suspicious packet arrival p:
1: **if** p matches an existing entry in H **then**
2: **if** $H_f.tResp \neq$ `null` **then** ▷ install pending
3: Drop p and **return**
4: **else if** $p.dir = H_f.dir$ **then** ▷ same-direction duplicate
5: Drop p and **return**
6: **else** ▷ reverse-direction (response) packet
7: Install a benign rule to the switch
8: $p_d^* \leftarrow$ `null`
9: Delete H_f from hash table H
10: **end if**
11: **else** ▷ suspicious first packet
12: Create packet descriptor p_d
13: $p_d.tArr \leftarrow$ `currentTime()`
14: Buffer packet p
15: $p_d.p^* \leftarrow addr(p)$ ▷ pointer to buffered packet
16: Insert flow entry H_f in H
17: $H_f.p_d^* \leftarrow addr(p_d)$ ▷ pointer to packet descriptor
18: $p_d.H_f \leftarrow H_f$ ▷ key in hash table entry
19: Enqueue p_d at tail of Q
20: **end if**

Periodic timeout check (every P_{DT}):
21: $depth \leftarrow 0$
22: $p_d^* \leftarrow$ head of Q ▷ oldest descriptor
23: $now \leftarrow$ `currentTime()`
24: **while** $p_d^* \neq$ `null` **and** $depth < d_{\max}$ **do**
25: **if** $now - p_d^*.tArr >$ DT timeout **then**
26: Delete entry H_f from hash table H
27: Forward buffered packet p to collector
28: Dequeue and free p_d^*
29: $p_d^* \leftarrow$ next descriptor in Q
30: $depth \leftarrow depth + 1$
31: **else**
32: **break** ▷ subsequent entries not yet expired
33: **end if**
34: **end while**

We describe how we balance the use of these two channels to filter benign traffic while keeping the switch rule set manageable.

1) Slow path: filtering long-lived benign flows with MAT rules: in production networks, long-lived internal services (e.g., web services) always respond to inbound packets and consistently generate well-formed bidirectional traffic, i.e., benign flows. CHAMALEONET identifies long-lived benign services and installs persistent filtering rules on the P4 switch using MATs. Because of stability and predictability, flows belonging to long-lived services are ideal candidates for the slow-path channel: since MATs memory has limited capacity and longer installation latency, we reserve MAT rules for filtering stable and well-known services.

The SDN control plane manages both rule installation and

removal. When pkt_rx threads identify flows belonging to these services as benign, they trigger rule installation via the SDN control plane, which installs exact-match MAT rules keyed by `(internal IP, internal port, protocol)`. Due to limited MAT memory capacity, the control plane also implements rule eviction to delete idle entries for (likely) terminated flows. We implement the entry eviction based on the idle notification mechanism provided by Intel Tofino chipset. Each MAT entry maintains a `TTL` field that decrements when no packet matches within a `Query_interval`; any matching packet resets the `TTL`. When the `TTL` reaches zero, the switch triggers an idle timeout notification to the controller, which also resets the corresponding host's aliveness status. We employ batching both for installation and removal to reduce overhead.

2) Fast path: filtering transient benign flows with Bloom filters: The fast-path is designed to cope with periods of high benign flow churn, where many new flows are initiated within short time intervals. In such scenarios, the switch would need to install numerous rules in a short time, which can outpace the speed of MAT rule installation over the slow-path channel and the MAT memory capacity. A Bloom filter [33] is a space-efficient probabilistic data structure that tracks membership in unbounded streams with constant-time query complexity, amenable for implementation on programmable switches. In CHAMALEONET, Bloom filters allow to (1) quickly install rules directly in the data plane without involving the control plane, and (2) filter the bulk of flows with a small memory footprint, albeit with some false positives whose impact can be made negligible (Sec. VI-D).

Standard Bloom filters do not support deletion or stale entry removal, which is necessary for CHAMALEONET to remove stale benign flows and detect internal hosts that go dark. To address this limitation, we adopt a rotating Bloom filter design that maintains r identical Bloom filters. Insertions occur only in the currently active Bloom filter, while queries span all r filters to determine whether a flow has appeared within the current time window or in any of the past $r-1$ time windows. Every T_{bloom}, CHAMALEONET rotates the active Bloom filter and clears the *oldest* one, thereby removing stale entries after at most $r \cdot T_{\text{bloom}}$. Consequently, benign rules stored in the Bloom filters have a maximum lifetime of $r \cdot T_{\text{bloom}}$, after which packets associated with these rules are treated as new suspicious packets and forwarded to FSD-NF for re-evaluation. By design, this overhead is negligible: transient flows typically complete before the rotation window expires, and persistent services are promoted to MAT rules via the slow path.

We favor rotation over alternative solutions, such as counting Bloom filters [34], because the latter would require the controller to maintain per-flow state to track staleness and issue explicit deletions.

3) Promotion of long-lived flows to MAT rules: we consolidate flow rules for long-lived benign flow into the slow path based on the flow lifetime. If a flow appears in n_{longlive} consecutive Bloom filter's clearing events—i.e., the benign flow lifetime is at least $n_{\text{longlived}} \cdot r \cdot T_{\text{bloom}}$—the corresponding service is identified as long-lived and the control plane migrates its flow rule from the fast path to the slow-path by installing an exact-match MAT rule. To track the number of consecutive Bloom filter rotations in which a flow f appears, the control plane increases a counter every time it receives a new suspicious packet for the same flow, after the first fast-path rule for f has been installed in the Bloom filter. If the counter reaches n_{longlive}, the control plane promotes the flow rule to the MAT and resets the counter.

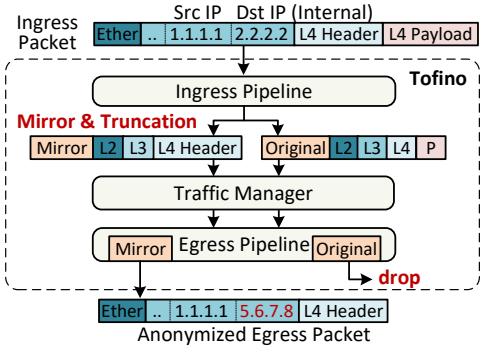


Fig. 5: P4-based traffic anonymiser implemented on a Intel Tofino [38] programmable chipset.

4) Filtering predefined benign services: To further reduce FSD-NF's load, we support a *static* rule set that the network administrator can compile to let the switch filter all *known* benign flows related to known services, in the form `dst IP, port`. For instance, this set can include well-known and benign services frequently visited by internal clients, e.g., popular cloud or CDN services. These rules are added via the slow-path only once during initialization and avoid unnecessary overhead in the FSD-NF and in the control plane to install per-flow rules for well-known benign services. Populating this list with the top-400 most popular external services like CDNs and video servers, we whitelisted up to 30% of traffic.

C. Traffic anonymizer

Privacy regulations such as GDPR [35], CCPA [36] consider "personal information" any information which can identify a person via an ID number, and information specific to the person. IP addresses are considered personal IDs, and the application layer payload may contain personal information. Given CHAMALEONET processes traffic generated by people connected to the campus network, we must adhere to the privacy regulations and access only the information we need for the specific task.[4] For this, CHAMALEONET supports configurable levels of anonymisation, directly implemented in P4 language for scalability. It supports internal IP address obfuscation and transport layer (L4) payload removal.

We leverage the Tofino Native Architecture [37], [38]: Packets first traverse the ingress pipeline before being buffered at the Traffic Manager (TM) and processed at the egress pipeline. Packet processing may happen at the ingress and egress stages, with the possibility to add metadata to coordinate the processing.

[4]Privacy policies have been validated by our Data Protection Officer (DPO).

For IP address obfuscation of internal hosts, we rely on consistent obfuscation by xor-ing the original value with a predetermined key salted with the last byte of the address itself. More complex techniques can be adopted [39].

Conversely, supporting L4 anonymisation through packet truncation requires some ingenuity. We leverage mirroring sessions, as shown in Fig. 5. For each packet to be truncated, mirroring permits the creation of a packet copy up to a specified length. We use ingress-to-egress mirroring mode to create such a packet copy at the ingress pipeline. The P4 program assigns a mirror `session_id` for each packet. When the ingress pipeline parses the original packet, it creates a copy with attached the `session_id` metadata header. The SDN controller configures the input port with a mirror session, sets the output port of this mirror session to the switch port connected to it, and configures the output of the original packet as DROP. When the mirrored packet arrives at the traffic manager of the Tofino chipset, it uses the `session_id` metadata to truncate the packet to the desired length. Next, the egress pipeline receives both the original and the truncated version of the packet. We forward the former to the output port and drop the original one. To distinguish the two replicas, we add an internal header field, denoted as `Mirror` and `Original` in Fig. 5, which is not serialized in the egress packet by the egress deparser.

To truncate packets differently for different protocols, we define the truncation sizes either by matching the correct length in the protocol headers or by assigning pre-defined values (e.g., for unknown/unsupported protocols), each matching a different mirror session.

V. ChamaleoNet in action

Testbed. We implemented the ChamaleoNet switch functionalities using 642 lines of $P4_{16}$ [26] code, using C/C++ and Python to implement the FSD-NF and controller functionalities. We deploy ChamaleoNet on the SUP4RNET [40] testbed, on a Debian 12 server equipped with 2 Intel Xeon Gold 6252N 24-core CPUs and 128 GB of RAM, connected to the P4 switch equipped with 6.5 Tbps Wedge 100BF-32X Intel Tofino ASIC [38]. We run FSD-NF and the SDN controller on the same virtual machine (VM) featuring 16 logic cores and 16 GB RAM.

Network configuration. We set the capture buffer size of each capture thread to 2 GB (the maximum libpcap buffer size[5]) to handle traffic bursts and periods in which the capture thread is stopped. The controller leverages the Barefoot Runtime Interface (BRI) to program the Tofino switch and communicates via a gRPC [41] channel. We set the batch size for rule installation and eviction to 200, which we found offering the best trade-off between gRPC overhead and rule installation latency on the Tofino chipset. Other parameters, such as the DT timeout and Bloom filter rotation interval, are discussed in Sec. VI-A.

We have deployed ChamaleoNet in a campus network for more than a year. The network is connected to the Internet with a 20 Gbps bidirectional link. At peak time, the border router forwards around 16 Gbps of total traffic. ChamaleoNet has collected data involving 34,304 internal IPv4 addresses.[6] Most of these addresses are allocated to desktops and laptops that go intermittently offline. NAT is heavily used for both the WiFi and some department networks. Since ChamaleoNet observes traffic after NAT translation, outbound analysis reflects NAT-translated source IP addresses rather than individual hosts, while inbound erroneous traffic remains unaffected as scanners target publicly routable addresses. Several campus servers offer regular services on the Internet. A border firewall protects all internal hosts, allowing incoming connections only to a subset of servers and services. As such, most internal hosts and services look unreachable from remote hosts. At last, a /23 subnet serves as a traditional telescope. It has been operating for more than 8 years to observe only Internet radiation.

Evaluation metrics and use cases. In this section, we report quantitative results obtained from the collected traffic traces. We consider the following metrics:

- the number of erroneous packets received;
- the number of unique external senders;
- the number of unique internal addresses contacted;
- the diversity of source ports used and destination ports targeted.

We compare these metrics between ChamaleoNet and a /23 dark address space (pure telescope), to quantify the additional visibility gained beyond traditional telescope-based monitoring. Unless otherwise stated, we aggregate statistics over 1-hour windows. Beyond aggregate metrics, we show how these traces translate into actionable operational insights through representative incidents ChamaleoNet has observed in the wild, including amplification attacks, external routing anomalies, service misconfigurations (see Sec. V-A), and compromised internal hosts, stale DNS configurations (see Sec. V-B).

Traffic filtering. In our deployment, ChamaleoNet's filtering mechanism effectively suppresses over 90% of the traffic, with the FSD-NF managing just 10% of the total traffic.

A. External Erroneous Radiation

ChamaleoNet offers visibility on external erroneous traffic, behaving as a *hybrid* telescope that mixes *dark* and *live* hosts. Here we investigate whether and how the addresses monitored by ChamaleoNet attract different traffic compared to the pure telescope.

Experiment setup: post-hoc analysis across live and dark hosts. For this study, we sample 25 days of data collected from December 7 to 31, 2024. To differentiate erroneous traffic targeting live versus dark hosts in ChamaleoNet, we track internal host liveness using a bitmap indexed by internal IP addresses; each bit represents the status of an internal host (live: 1, dark: 0). An internal host is marked *live* when we observe an outbound packet from it, either filtered by the P4 switch as part of an ongoing flow or mirrored to FSD-NF when no matching rule exists. FSD-NF embeds the liveness status

[5]https://www.tcpdump.org/manpages/pcap_set_buffer_size.3pcap.html

[6]The campus uses one /17 and six /24 subnets, summing 34,304 addresses.

Probes	Dst. IPs	Dst. Ports	Src. ASs	Src. IPs	Acknowledged Scanners
/23 telescope	510	1k	715	7k	708
CHAMALEONET	34304	62k	2310	17k	935

TABLE II: Quantitive comparison between traffic collected by CHAMALEONET and pure telescope every hour.

(one bit per packet) into a customized header field, letting the erroneous packets carry per-packet metadata to the collector. This mechanism enables post hoc analysis across live and dark destinations, as shown in the following measurements.

Sender diversity and coverage. In general, CHAMALEONET observes on average 17215 unique external senders and 1.21×10^7 erroneous flows. Table II shows the average amount of external erroneous traffic observed every hour. CHAMALEONET captures traffic from substantially more senders: 3.2× more unique Autonomous System (AS), 2.4× more unique IP addresses, and 1.3× more acknowledged scanners.[7] This sub-linear growth is expected because many large-scale scanning campaigns probe broad portions of the IPv4 space, making most scanners already visible in the /23 deployment. Expanding coverage therefore primarily increases the interaction intensity per scanner, while newly observed scanners are mainly from small, localized probing activities (see Fig. 6 in the following). Yet, senders in CHAMALEONET are also significantly more aggressive, contacting almost the whole port range (62×).

Sender aggressiveness. We analyze the distribution of traffic statistics in Fig. 6, including (a) packets, (b) distinct source IPs, (c) distinct source ports, and (d) distinct destination ports. To ensure a fair comparison between CHAMALEONET (/17) and the /23 telescope, all metrics are computed on a per-destination-IP basis.

Given the fact the most of addresses monitored by CHAMALEONET is *dark* in practice, we expect their traffic statistics to be similar to the pure telescope, as reflected by the overlapping CCDF curves. However, CHAMALEONET exhibits consistently heavier tails across all four metrics: the top 0.1% of destinations receive substantially more packets (up to 14×), from more distinct senders (up to 11×), with greater source-port diversity (up to 8×) and broader destination-port coverage (up to 3×). Fig. 6(b) reveals stronger concentration effects: certain hosts in CHAMALEONET are contacted by significantly more unique senders. Fig. 6(c) indicates more aggressive probing behaviors, consistent with scanners randomizing source ports or issuing repeated connection attempts. Fig. 6(d) suggests that some destinations are subjected to wider port coverage, attributing to deeper or more comprehensive scanning activity. Overall, erroneous traffic collected by CHAMALEONET exhibits consistently heavier-tailed per-destination distributions, showing increased scanner aggressiveness in terms of traffic volume and probing breadth.

Fig. 7 shows the number of packets per port observed in the top three Telescope receivers (top), active receivers (middle), and dark receivers (bottom) within CHAMALEONET over one day. The figure clearly shows that external senders target different port ranges with different intensities. While well-known ports and low port numbers receive more radiation, the presence of active services causes a significant increase in unsolicited traffic, causing the per-port probability to change.

[7]Ground truth: https://gitlab.com/mcollins_at_isi/acknowledged_scanners

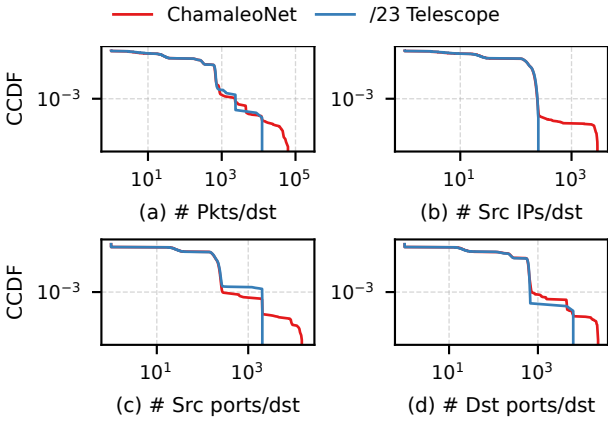


Fig. 6: Distribution of Traffic Statistics (CCDF, Log–Log Scale). (a) # Pkts/dst, (b) # Src IPs/dst, (c) # Src ports/dst, and (d) # Dst ports/dst denote the number of packets, unique source IPs, unique source ports, and unique destination ports observed per destination IP address, respectively.

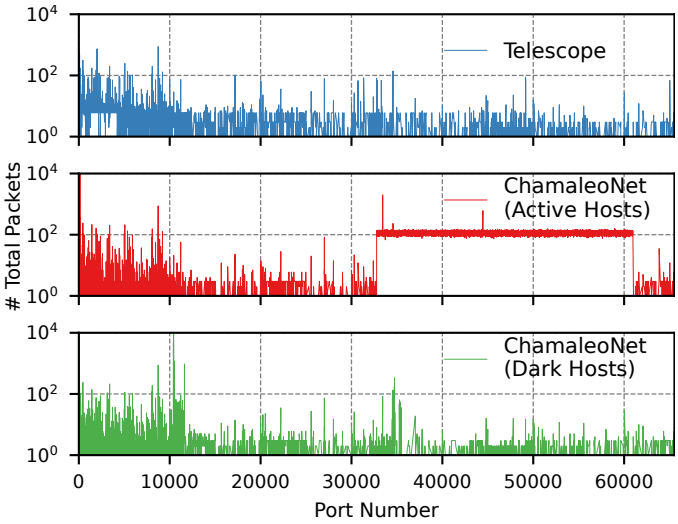


Fig. 7: Erroneous packets per destination port on the telescope, active hosts and dark hosts.

In a nutshell, CHAMALEONET collects much more Internet radiation than a pure telescope. As [8], [9] transformed a CDN and a cloud provider into a telescope, CHAMALEONET transforms any live network into a better telescope, without the need to reserve precious IP addresses for this.

Temporal evolution. We now examine the temporal evolution of external erroneous radiation using two representative /24 subnets monitored by CHAMALEONET. *ServerNet* is a /24 subnet hosting 10-15 campus servers, the other addresses acting as dark hosts. *UserNet* is a /24 subnet with client hosts that are unreachable from outside. For comparison, we also include one /24 telescope subnet. Fig. 8 illustrates the erroneous packets for each address in *ServerNet*, *UserNet*, and *Telescope*, separated by solid blue lines.

Scan pattern is very similar, with different intensities – notice the 48-hour intensive scanning event during Dec. 23th-

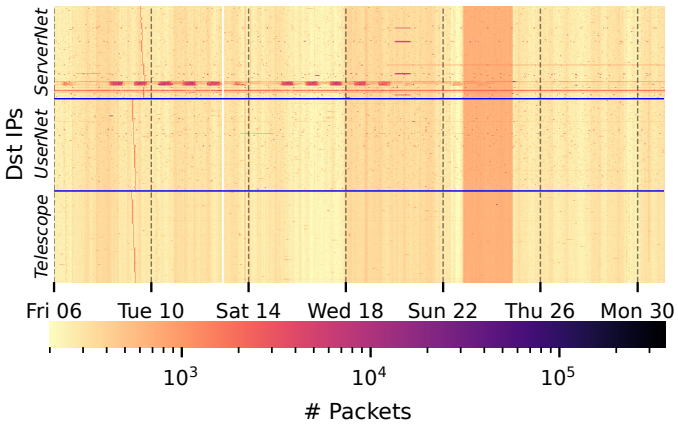


Fig. 8: Incoming traffic collected by CHAMALEONET and pure telescope (1-hour interval). Blue lines mark the three /24 subnets.

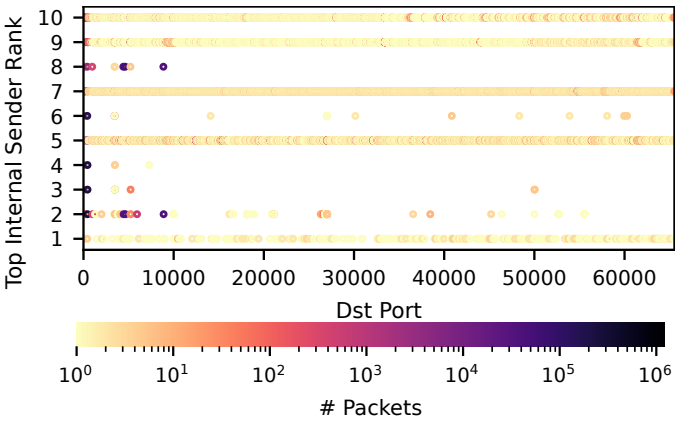


Fig. 9: TCP destination ports of internal erroneous traffic generated by the top-10 contributors.

24th period. Yet, the *ServerNet* shows clearly some different pattern: the continuous horizontal line refers to active servers that external actors keep scanning on several ports. Some intermittent pattern related to human activity emerges. Here, thousands of external addresses send erroneous packets to the public addresses allocated to the campus WiFi network NAT. These packets are due to ongoing connections where the internal WiFi host results suddenly unreachable, and the external senders keep retransmitting packets. At last, in the *ServerNet* we observe some sudden and prolonged increase in erroneous traffic (small red horizontal bars). These are external scanners that look for other servers in this /24 subnet.

External senders targeting campus addresses. During the study period, we observed a total of 436k unique external senders. Among them, 56.7% exclusively target campus addresses and avoid any telescope address (while only 0.3% interact solely with the telescope). By analysing the top-senders, we identify a variety of behaviours, including attack patterns, scanning activities, misconfigurations, etc. We describe some notable findings below (with anonymized addresses).

- IP 5.96.X.X: This is the most active sender. It generates an average of 3,000 packets per hour for the entire period. All packets go to UDP port 123 on a specific internal server, strongly suggesting a Network Time Protocol (NTP) attack.
- IP 188.92.X.X: It scans ports 37215, 52869, and 49152 on thousands of hosts, hinting at a Satori bot [42]. It also scans other ports (e.g., 1900 and 2048) indicating broader scanning. The sender is present in blocklists.
- IP 193.X.X.X: It belongs to the campus network provider. It keeps sending more than 1,000 ICMP Time Exceeded messages per hour, targeting 336 internal addresses. This suggests a routing issue (or blackholed destination). Internal clients cannot reach the final destination.
- IP 94.143.X.X: It consistently contacts port of a internal server from source port 57498, sending about 720 packets per hour. This indicates a misconfigured or failed service.
- 38 IP addresses (e.g., 49.232.X.X) conduct synchronised ICMP scans throughout the period, sending an average of 394 ICMP Echo Requests per hour to 20 internal IP addresses. The campus firewall blocks ICMP echo requests.

Firing impersonators to engage with scanners. To understand which services the scanners are interested in, we activate TCP impersonator on some unused IP addresses on the campus for three hours. These impersonators complete the TCP three-way handshake, observe the first application message, if any, and tear down the connection. For impersonation traffic, CHAMALEONET operates in a pass-through mode with explicit redirection. The switch bypasses filtering and anonymization and forwards traffic to FSD-NF directly. In this mode, FSD-NF disables buffering and detection logic and forwards traffic to the responders, while simultaneously mirroring a copy to the collector for deeper analysis. We run nDPI [2] to extract the application protocol classification of captured traffic, focusing on traffic to ports in the $[30000, 61000]$ range. Results show a mix of different protocols, the majority unknown to nDPI (45%) (because only the first payload is captured), or has no payload (21%) (because of client-initiated protocol), with HTTP (18%), TLS (4%), LDAP, DRDA, DRP, SQL, PRTP and other protocols being probed (<1%). This confirms that the external scanners probe for various services on non-standard ports once they find the host alive [19]. The impersonating ability of CHAMALEONET is fundamental for exploring the intention of senders.

B. Internal Erroneous Radiation

Let us focus on the erroneous packets generated by internal hosts that CHAMALEONET captures. Recall that a pure telescope will not collect any information on this. On average, we observe approximately 40 k outgoing erroneous packets per hour, consisting of roughly 40% TCP packets, 36% UDP packets, and 24% ICMP packets. Again, we use simple analytics to quickly highlight suspicious behaviours.

Top senders of internal erroneous traffic. We look at those hosts that contribute the most to the erroneous radiation. We consider one day of traffic. Each of the top senders contributes tens of thousands of erroneous packets. Fig. 9 shows the destination ports targeted by the top-10 senders, the most active on the bottom. Two patterns emerge: (i) horizontal scan patterns that target all (e.g., 1st, 4th-6th senders) or most ports (e.g., 9th sender); (ii) vertical scan patterns, targeting few ports (e.g., 3rd, 7th senders). Some vertical scanners constantly send thousands of unanswered requests per hour

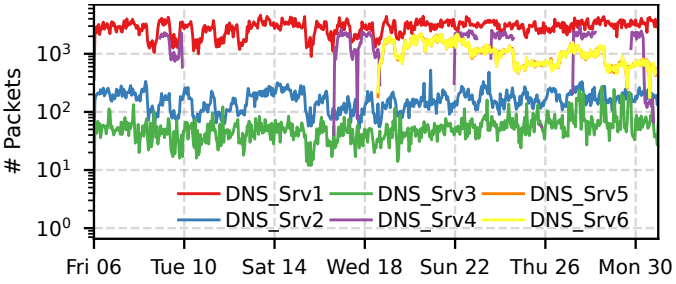


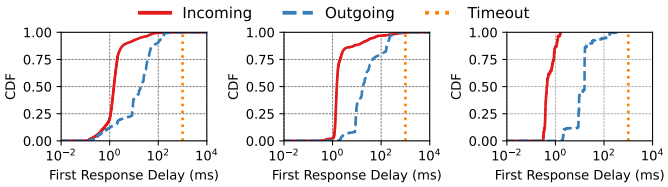
Fig. 10: Erroneous DNS traffic sent by the campus DNS servers to external DNS servers.

to very few IP addresses in a cloud system. This hints at misconfigured systems that keep sending traffic to non-existent servers (some examples later). For horizontal scanners, some target a few external hosts while others contact hundreds of different destinations. This hints at bots or malicious scanning activities that we signalled to our IT security team.

Erroneous DNS traffic. By looking at the most targeted ports, we notice a lot of erroneous UDP traffic destined to port 53/DNS and sent by the campus official DNS resolvers. Digging into this, we observe that most of this traffic is directed to a few authoritative DNS resolvers (in Iran, China, India), which never responded to our resolvers. We report the temporal evolution of this erroneous traffic in Fig. 10. We suspect this suspicious activity is part of attacks where infected internal clients send DNS requests to our resolvers that route them to the authoritative resolvers. Again, we signalled the incident to our IT security team.

Other findings. Among the recurrent patterns CHAMALEONET exposes, we notice some seldom peaks of unanswered traffic going to popular external servers. Unlike the above cases that show explicit malicious intent, these patterns originate from benign communications yet exhibit erroneous behaviour, potentially caused by misconfigurations or transient operational issues. Importantly, our objective is not attack detection per se, but operational visibility. Therefore, such events are treated as visibility signals reflecting abnormal network conditions, rather than classification errors within a malicious traffic detection framework. For instance, we observe a steady stream of TCP SYN packets – approximately 3.7k per hour – originating from a specific internal server and directed at a remote HPC repository. Upon investigation, we discovered that the remote repository had been decommissioned, but its DNS name remained registered. As a result, the internal server continued attempting to reconnect to it. Another notable case involves more than 50 internal hosts generating unusual ICMP port unreachable messages. Upon investigation, we identify that these messages were triggered by aggressive DNS lookup behaviour. Several of the affected hosts were confirmed to be running DNS acceleration tools. These DNS accelerators improve resolution speed by sending DNS queries in parallel to multiple open resolvers. The host accepts the first response it receives and immediately closes the corresponding socket. When subsequent (late) responses arrive, the socket is already closed, causing the system to generate ICMP port unreachable messages, appearing as erroneous traffic to CHAMALEONET.

Understanding the precise reasons for these behaviors is outside our scope. However, these simple findings illustrate the benefits of the visibility CHAMALEONET provides.



(a) TCP **(b) UDP** **(c) ICMP**

Fig. 11: First response delay of incoming and outgoing flows, measured in our campus network.

VI. SYSTEM PERFORMANCE EVALUATION

We now present results collected in controlled experiments to optimize CHAMALEONET design, select system parameters and characterize scalability.

A. Parameter Setting

Setup. We set up a second VM to run a traffic generator, bridged to the server E810 NIC in PCIe passthrough mode using a separate SR-IOV [43] virtual function. Packets transmitted from the traffic generator VM are forwarded to the P4 switch before returning to the FSD-NF VM. We run a traffic replay workload that uses realistic traffic patterns captured from a campus network, containing about 6 million flows and 152 million packets over 10 minutes. We use up to 10 `tcpreplay` [44] instances to generate different traffic rates, splitting and multiplexing traces in parallel to mimic real traffic patterns. As regards the fast-path configuration in the P4 switch data plane, our implementation uses r=2 Bloom filters, each with 2 hash functions and a bit array of 2^{22} entries. We set these parameters to the largest values allowed by the compiler and per-stage resource constraints of Tofino. Finally, we set T_{bloom} = 5s for bloom filter rotation and $n_{\text{longlived}}$ = 2 for the long-lived flow promotion (Sec. IV-B3).

DeTection (DT) timeout. For setting the DT timeout we observe the empirical response delay statistics of actual flows. We analyse the campus traces to measure the *first* response delay of answered flow, i.e., the time elapsed between the first packet and the first response packet. We observe more than 100,000, 72,000 and 1,000 TCP, UDP and ICMP flows, respectively. About one-tenth are incoming, i.e., requests launched by external clients targeting internal services.

Fig. 11 shows the Cumulative Distribution Function (CDF) of the response delay for TCP, UDP, and ICMP, respectively. The CDFs show that about 80% of incoming requests are answered in a few ms. This is expected since internal hosts located within the main campus LAN have a short Round Trip Time (RTT). Because our campus also includes some remote laboratories connected with VPNs over the Internet, some clients and servers suffer longer RTT. Looking instead at outgoing flows, the response time is significantly higher as it includes the RTT to servers located anywhere on the Internet.

The choice of a proper DT timeout is a balance between accuracy and system load: increasing the timeout improves flow identification accuracy, but impacts system load and

scalability. In our setting, we set the timeout to 1s to correctly classify over 99.9% incoming and 99.8% outgoing answered flows. Note that CHAMALEONET could be easily extended to support multiple timeouts by simply splitting traffic to separate NFs or packet descriptor queues, e.g., using different DTs for incoming and outgoing requests, as done in telescope-like systems [45], [18], [9].

DT timeout expiration routine: tuning (P_{DT}, $d_{\max}$). As elaborated in Sec. IV, we adopt the same thread pkt_rx both to process incoming suspicious packets and manage DT timeout expiration. Every P_{DT} seconds, we halt the packet reception routine to check if the timer for the HoL descriptor has expired and start popping entries from the ring queue until the expiration time is smaller than DT or we reach the maximum scanning depth $d_{\max}$. The time CHAMALEONET spends in the timer check routine directly affects the per-packet processing time, with incoming packets that are buffered at the `libpcap` buffer while we halt pkt_rx for a while. Notice that the buffering delay does not alter the system functionality as long as no packet is dropped.

We measure the per-packet processing time of the controller by computing the elapsed time from when it arrives at the controller until we finish its processing. We sample 1 out of $10,000$ packets to not alter the regular processing.

Under 10x `tcpreplay` senders load, we configure the system to perform timeout checks at a rate of $d_{\max}/P_{DT} = 1$ million per second, and repeat the experiment three times to mitigate randomness. We show results in the top plot of Fig. 12 for different combinations of P_{DT} and $d_{\max}$ and for the 75, 95 and 99 percentiles. CHAMALEONET processes the majority of packets within a relatively short time frame. Notably, we observe that dividing timeout checking into more frequent, smaller-scale operations significantly improves per-packet performance. Choosing $P_{DT} = 0.01$ ms and $d_{\max} = 10$ provides the best tradeoff. Specifically, 99% of packets are processed within 3 ms – with no packet loss, compared to 55 ms when checking up to 10,000 descriptors every 10 ms. For completeness, the rightmost plot shows the packet processing time distribution under real traffic trace. The burstiness of real traffic even improves the overall results.

Let us now focus on the amount of time suspicious packets are buffered at the FSD-NF (bottom plot of Fig. 12). Ideally, packets should stay in the buffer for DT so that when the timer expires, they are removed and sent to the collectors. However, the lazy processing of the timer may cause packets to be buffered longer than DT. To gauge this, we measure the per-packet buffering time under different timeout-checking settings. The configuration with $P_{DT} = 0.1$ ms and $d_{\max} = 100$ achieves the best performance, with approximately 99% of packets being buffered for 1.005 seconds, showing negligible deviation from the predefined DT timeout. This observation is further supported by the results from real trace data.

In general, splitting timeout checks into more frequent, smaller-scale operations improves per-packet performance and precision in managing DT timeouts. The above results reveal a broad stable operating region for $(P_{DT}, d_{\max})$. We therefore adopt empirically chosen fixed parameters within this region.

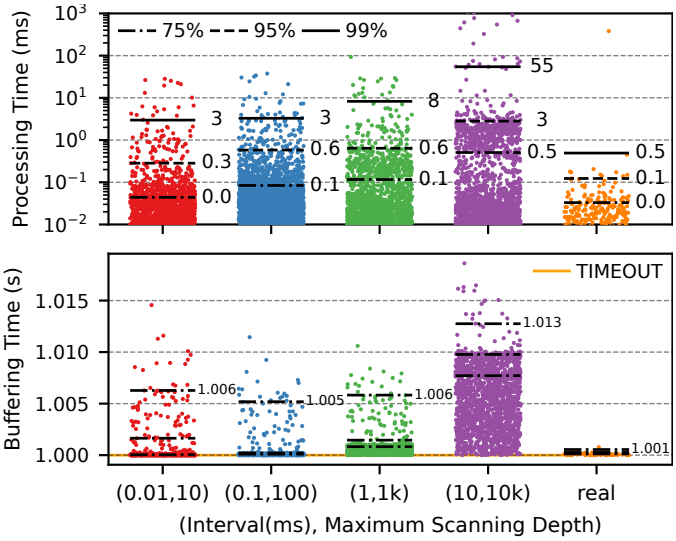


Fig. 12: Periodic check of timeout expiration within pkt_rx thread for various configurations of $(P_{DT}, d_{\max})$, with DT timeout set to $1s$. "Real" refers to the deployment in our campus network.

B. Scalability Analysis

Traffic Workloads. We evaluate CHAMALEONET's performance using two traffic workloads that stress different system components.

- *DoS traffic*. This synthetic workload stresses FSD-NF's buffer capacity and processing throughput by maximizing concurrent erroneous flows. Each packet forms a distinct flow and is classified as erroneous. We use the Cisco T-Rex [46] traffic generator built on DPDK to generate this workload.
- *High-churn benign traffic* This synthetic workload stresses rule management by rapidly generating benign flows. Each flow consists of exactly two bidirectional packets and is therefore classified as benign, testing the capacity of benign rules and the efficiency of rule updates. We also use T-Rex to generate this workload.

Throughput under DoS workloads. DoS attacks or horizontal scanning attacks represent worst case scenarios for CHAMALEONET's scalability, as all incoming packets correspond to a new flow and translate into large bursts of suspicious packets for the FSD-NF. We evaluate maximum throughput following the benchmarking methodology specified in RFC 2544 [47].[8] Fig. 13 reports the achieved throughput with one and two pkt_rx threads under different packet buffer sizes. A single pkt_rx thread achieves a peak throughput of approximately 350 kpps, whereas two threads achieve up to 840 kpps, corresponding to a 2.4× improvement. This super-linear gain likely results from increased memory-level parallelism under multi-core execution, while hash-based flow distribution further improves workload uniformity across threads. When the buffer is smaller than 350 kpps, insufficient memory prevents newly arrived packets from being buffered, leading to early overflow. As the buffer size increases, packet processing operations (e.g., buffering, expiry checking, and

[8]We stopped the DDoS upon the first packet loss at the FSD-NF and measured peak throughput.

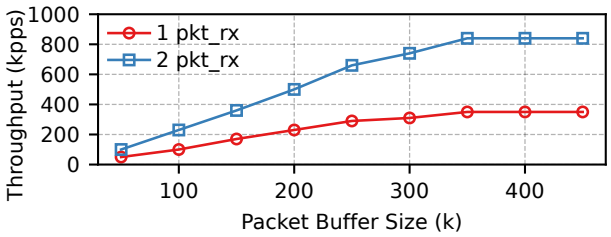


Fig. 13: Throughput of one and two pkt_rx threads under DoS workload for different packet buffer sizes.

garbage collection) become the dominant bottleneck. In this case, packets cannot be drained from the libpcap ring buffer fast enough, resulting in packet drops. These results confirm that CHAMALEONET achieves scalable throughput by parallelizing packet processing across multiple pkt_rx threads.

Scaling to the peak load caused by a DoS attack is not an imperative requirement for CHAMALEONET, which focuses on enhancing visibility and may simply drop excess traffic at the FSD-NF. – e.g., by detecting and stopping the DoS in the switch itself [48], [49], [50], [51]. DoS resilience is a common challenge for monitoring systems in general. For example, Zeek [13] notes that generating a `new_connection` event for every incoming connection can become expensive under SYN flooding attacks. Similar to existing solutions, CHAMALEONET can integrate switch-level packet processing mechanisms to detect and suppress DoS traffic [52], [50], [51]. However, the design and evaluation of DoS mitigation mechanisms are orthogonal to the focus of this work.

Throughput under high-churn workload. We also evaluate CHAMALEONET under high-flow-churn workloads. With two pkt_rx threads, throughput also reaches 840 kpps (i.e., 420 k flows per second), bounded by packet processing capacity. Fast-path rule installation into the Bloom filter does not introduce scalability concerns. In contrast, slow-path updates may become a bottleneck due to the limited processing capacity of cnt_tr, which dequeues entries from the shared socket across FSD-NF instances, and installs them into the switch via gRPC channel. Due to control-plane bandwidth limitations on Tofino, our measurements show that it can sustain at most approximately 14k rule updates per second. When the slow path update rate exceeds this capacity, cnt_tr applies random sampling to limit long-lived rule promotion. This mitigation trades promotion accuracy for scalability, while fast-path filtering continues to effectively handle benign traffic.

C. Resource Usage

Switch resource usage. First we present the scalability analysis of the switch resource usage, under high-flow-churn, replay, and real workloads. We study the impact of the Bloom filter rotation interval T_{bloom} on the number of entries in the Bloom filter and the MAT. The number of entries in the Bloom filter affects CHAMALEONET's false negative rate (analysis in Sec. VI-D), while the number of MAT entries reflects the number of long-lived services that are promoted to exact-match rules, which directly impacts the TCAM usage.

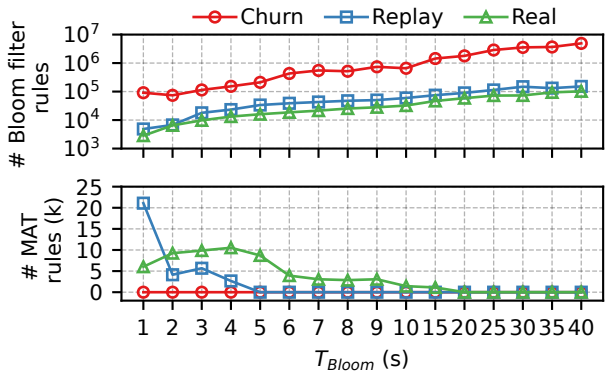


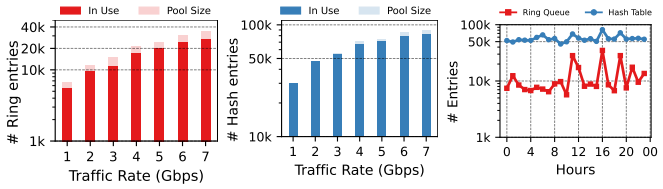
Fig. 14: Number of Bloom filter rules (top) and MAT rules (bottom), for different time windows T_{bloom}. Every T_{bloom}, CHAMALEONET rotates the active Bloom filter and removes entries from the oldest one.

Fig. 14 (top) quantifies the growth in the number of rules in the Bloom filters. High-churn traffic exhibits more intense use of the Bloom filter memory, with the number of entries ranging from 9.1×10^5 to 4.9×10^6, since each flow triggers an insertion. In contrast, replay and real workloads exhibit substantially lower Bloom filter occupancy due to higher flow stability and lower churn, ranging from 4.86×10^3 to 1.51×10^5 and from 2.83×10^3 to 1.02×10^5, respectively.

The bottom figure shows the number of MAT entries. Since long-lived promotion requires a service to appear across multiple Bloom rotation epochs ($n_{\text{longlived}} = 2$ in our setting), increasing the rotation interval T_{bloom} makes promotion more stringent. Note that for high-churn traffic, no flows are promoted because flows do not persist long enough to satisfy this criterion. For real workloads, MAT entries first increase and then decrease as the rotation interval grows. Very short rotation intervals (e.g., 1s) reduce the likelihood that a persistent service is re-observed within each consecutive epoch, causing fewer flows to satisfy the consecutive-appearance criterion for promotion. Slightly longer intervals (2 – 3s) increase the probability that persistent services appear in every epoch and thus get promoted. The unusually high MAT count at 1s under replay workload is caused by multiplexed replay, which concentrates packets of the same flow within short time spans. As the interval further increases, longer Bloom retention reduces re-observations of benign flows, lowering promotion frequency. Beyond 5s (replay) and 20s (real), no services satisfy the long-lived condition, and MAT entries drop to zero.

Regarding memory usage, according to the Tofino compiler report [37], the P4 program of CHAMALEONET consumes 26.8% SRAM, 42.0% Map RAM, and 0.7% TCAM resources.

FSD-NF data structure usage. As described in Sec. IV-A, FSD-NF relies on three data structures: the hash table, the packet descriptor queue and the packet buffer. To evaluate the resource utilization under different traffic loads, we periodically sample the amount of allocated memory pool and the currently in-use active entries of the ring queue and hash table. Fig. 15a and 15b show the usage of packet buffer (equal by construction to the descriptor queue) and hash table, respectively. Notably, we collect results at CHAMALEONET



(a) Ring queue usage **(b) Hash table usage** **(c) Real deployment**

Fig. 15: Resource utilization of FSD-NF for replayed traces (a)–(b) and real deployment (c).

Interval	1s	5s	10s	15s	20s	25s	30s	35s	40s
Churn	0.45	2.76	16.91	42.29	46.28	66.76	86.65	88.36	96.64
Replay	0.001	0.05	0.16	0.24	0.36	0.57	1.01	1.02	1.04
Real	0.00004	0.01	0.05	0.10	0.15	0.23	0.24	0.38	0.47

TABLE III: False negative rates (%) of CHAMALEONET's filtering for different rotation intervals under high-churn, replay, and real workloads.

with all filtering mechanisms enabled. As expected, with increasing load, the memory usage grows to 20k ring entries and 80k hash table entries. With this, we set the hash table size equal to 20M buckets, which guarantees collision rate below 0.5%, p99 lookup depth below 1.02 under real workload. For completeness, Fig. 15c reports these figures in the real deployment. Since the number of entries depends on the number of erroneous packets, some spikes may reflect the intensification of erroneous activities. All in all, the memory management does not pose particular challenges.

D. Impact of Bloom filter on filtering accuracy

Storing benign flow rules in Bloom filters may cause false negatives for CHAMALEONET: a false positive in any of the r Bloom filters can incorrectly match an erroneous packet to a benign flow entry, causing the packet to be discarded. However, the false negatives remain negligible in practice under the different workloads. To quantify, in Fig. 14 (top) we measure the Bloom filter occupancy in our experiment, i.e., the number of elements in the filter, and derive the false negative rate in Table III. The false negative rate is computed as FNR = $1 - (1 - \text{FPR}_{BF})^r$, where $\text{FPR}_{BF} = (1 - e^{-kn/m})^k$ is the standard Bloom filter false positive rate for k hash functions, n inserted elements, and m-bit array. Recall, our implementation uses k=2, m=2^{22}, and r=2. Under replay and real workloads, Bloom filter occupancy remains low, yielding FNR between 0.001%–0.97% and 0.0004%–0.45%, respectively. Under high-churn conditions FNR increases from 0.24% to 96.64% as the rotation interval grows from 1s to 40s. Importantly, false negatives only impact the completeness of erroneous traffic collection, not the correctness of benign flow classification nor the stability of the filtering mechanism. With appropriate rotation intervals, the FNR remains low under real workloads and does not affect the main conclusions of our analysis.

VII. DISCUSSION

Offloading FSD-NF to programmable dataplanes. CHAMALEONET leverages the synergy between a programmable data plane component and a software-based NF. Offloading the FSD-NF to the dataplane could realize CHAMALEONET entirely in the data plane, eliminating the need for a separate NF. However, there are at least two fundamental challenges in this design. First, buffering every suspicious packet for *tens of milliseconds* on constrained on-chip memory at today's link speeds remains infeasible. As an example, a 10MB buffer (in line with the available memory on a datacenter switch [53], [54]) would be filled up in about a millisecond at 100Gbps. Second, given the lack of random access memory on programmable switch ASICs, it is non-trivial to access a buffered suspicious packet once is time to discard it (Sec. IV-A). Exploring CHAMALEONET integration with state-of-the-art work in this space, such as PayloadPark [55], is left as a future enhancement.

In-path deployment. CHAMALEONET's architecture (Sec. III-A) relies on mirroring ingress/egress traffic from the campus network to the SDN switch, resulting in an out-of-path deployment. Alternatively, CHAMALEONET could be deployed in-path on SDN-capable border router (or any downstream switch on the ingress/egress path). In such cases, the FSD-NF shall observe all suspicious packets and install a rule to *forward* benign flow packets to the campus network instead of *dropping* them.

In-path deployment enables the defense from erroneous flows, i.e., dropping or re-routing erroneous packets instead of forwarding them to the campus network. However, in-path deployment introduces extra challenges, such as invertible anonymisation and additional forwarding latency during initial flow state detection. It is also more vulnerable to false negatives: missing the detection of a benign flow would directly disrupt normal traffic. Consequently, network administrators may be reluctant to introduce CHAMALEONET on the live traffic path. The out-of-path design illustrated in Fig. 2 offers a less intrusive and pluggable solution.

VIII. ETHICAL CONSIDERATIONS

Privacy. Privacy is fundamental as legitimate traffic is also mirrored. We consider an "honest but curious" scenario where the person in charge of processing the data will not deviate from the defined goals. We thus adopt a *data minimisation approach*: To minimise the information the controller and the collectors process and store, we keep only headers up to the transport protocol and remove any upper-layer information. We support IP address anonymisation for the internal network IP addresses by using a simple obfuscation mechanism. We leverage the data-plane programmability features to perform anonymisation directly at the switch (see Sec. IV-C).

Impersonating not-responding servers or services. CHAMALEONET cannot interfere with production traffic. Logging erroneous packets is not critical – impersonating non-responding servers or services requires particular attention. When internal services go offline for maintenance or temporary failures, legitimate users could find themselves interacting with honeypots. Likewise, impersonating an internal offline client (e.g., a PC turned off at night) poses legal and security concerns if its address is converted into honeypot. We limit ourselves to demonstrating the CHAMALEONET impersonating ability with some pre-determined off-line addresses,

leaving its extensive usage for future analysis. Notice that the privacy features must be disabled for the hosts/services CHAMALEONET will impersonate so that the original IP addresses and payload are exposed to the impersonator.

IX. CONCLUSIONS

We presented CHAMALEONET, a system that transforms any production network into a programmable probe to monitor erroneous traffic – such as unanswered, misrouted, or malformed packets. By leveraging SDN principles and programmable data planes, CHAMALEONET offers scalable, privacy-compliant traffic visibility without disrupting normal network operations. It operates transparently in live environments, collecting only erroneous traffic, anonymising sensitive information at the switch, and enabling both passive observation and active engagement through honeypot integration.

Our deployment over eight months showed CHAMALEONET effectiveness in uncovering external scanning behaviours and internal misconfigurations, outperforming traditional telescope setups in data richness and insights. It filtered up to 96% of benign traffic at the hardware level, ensuring scalability with minimal resource requirements. CHAMALEONET is open source, modular, and ready for deployment in various environments, offering a practical and ethical solution to enhance network visibility and threat intelligence without requiring dedicated addresses or additional instrumentation.

ACKNOWLEDGMENT

This work was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU and the PRIN ACRE (AI-Based Causality and Reasoning for Deceptive Assets – 2022EP2L7H). This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them.

REFERENCES

[1] M. Trevisan, A. Finamore, M. Mellia, M. Munafò, and D. Rossi, "DPDKStat: 40Gbps Statistical Traffic Analysis with Off-the-shelf hardware," Telecom, Paris, France, Rep, 318627, Tech. Rep., 2016.

[2] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi: Open-source high-speed deep packet inspection," in *IEEE IWCMC*, 2014, pp. 617–622.

[3] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "FloWatcher-DPDK: Lightweight line-rate flow-level monitoring in software," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 3, pp. 1143–1156, 2019.

[4] M. Ozkan-Okay, R. Samet, Ö. Aslan, and D. Gupta, "A comprehensive systematic literature review on intrusion detection systems," *IEEE Access*, vol. 9, pp. 157 727–157 760, 2021.

[5] O. H. Abdulganiyu, T. Ait Tchakoucht, and Y. K. Saheed, "A systematic literature review for network intrusion detection system (IDS)," *Int. J. Inf. Secur.*, vol. 22, pp. 1125–1162, 2023.

[6] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of Internet background radiation," in *ACM IMC*, 2004, pp. 27–40.

[7] D. Moore, C. Shannon, G. M. Voelker, S. Savage *et al.*, "Network telescopes," CSE Department, UCSD, USA, Rep. CS2004-0795, Tech. Rep., 2004.

[8] E. Pauley, P. Barford, and P. McDaniel, "DScope: A cloud-native Internet telescope," in *USENIX Security*, 2023, pp. 5989–6006.

[9] P. Richter and A. Berger, "Scanning the scanners: Sensing the internet from a massively distributed network telescope," in *ACM IMC*, 2019, pp. 144–157.

[10] D. Wagner, S. A. Ranadive, H. Griffioen, M. Kallitsis, A. Dainotti, G. Smaragdakis, and A. Feldmann, "How to operate a meta-telescope in your spare time," in *ACM IMC*, 2023, p. 328–343.

[11] A. Sordello, Z. Wang, K. Huang, A. Cornacchia, and M. Mellia, "Poster: The potential of erroneous outbound traffic analysis to unveil silent internal anomalies," in *Proceedings of the 2025 ACM Internet Measurement Conference*, ser. IMC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1078–1079. [Online]. Available: https://doi.org/10.1145/3730567.3768599

[12] D. Ergenç, R. Schenderlein, and M. Fischer, "Tsnzeek: An open-source intrusion detection system for ieee 802.1 time-sensitive networking," in *IFIP Networking*, 2023, pp. 1–6.

[13] Zeek Project, "Zeek documentation," Zeek.org. Accessed on Aug. 2025. [Online]. Available: https://docs.zeek.org/en/current/scripts/base/bif/event.bif.zeek.html#id-new_connection

[14] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, 2017.

[15] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the DoS ecosystem," in *ACM IMC*, 2017, pp. 100–113.

[16] M. Antonakakis *et al.*, "Understanding the Mirai Botnet," in *USENIX Security*, 2017, pp. 1093–1110.

[17] CAIDA, "The UCSD Network Telescope," Accessed on Apr. 2025, 2024. [Online]. Available: https://www.caida.org/projects/network_telescope/

[18] R. Hiesgen, M. Nawrocki, A. King, A. Dainotti, T. C. Schmidt, and M. Wählisch, "Spoki: Unveiling a new wave of scanners through a reactive network telescope," in *USENIX Security*, 2022, pp. 431–448.

[19] F. Soro, T. Favale, D. Giordano, I. Drago, T. Rescio, M. Mellia, Z. B. Houidi, and D. Rossi, "Enlightening the darknets: Augmenting darknet visibility with active probes," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, pp. 5012–5025, 2023.

[20] A. Waleed, A. F. Jamali, and A. Masood, "Which open-source IDS? snort, suricata or zeek," *Comput. Netw.*, vol. 213, p. 109116, 2022.

[21] P. Hadem, D. K. Saikia, and S. Moulik, "An SDN-based intrusion detection system using SVM with selective logging for IP traceback," *Comput. Netw.*, vol. 191, p. 108015, 2021.

[22] B. Ayodele and V. Buttigieg, "SDN as a defence mechanism: A comprehensive survey," *Int. J. Inf. Secur.*, vol. 23, pp. 141–185, 2024.

[23] R. Li, M. Zheng, D. Bai, and Z. Chen, "SDN based intelligent honeynet network model design and verification," in *MLISE*, 2021, pp. 59–64.

[24] M. Karakate, H. Esaki, and H. Ochiai, "SDNHive: A proof-of-concept SDN and honeypot system for defending against internal threats," in *ACM ICCNS*, 2022, p. 9–20.

[25] G. Li, M. Zhang, S. Wang, C. Liu, M. Xu, A. Chen, H. Hu, G. Gu, Q. Li, and J. Wu, "Enabling performant, flexible and cost-efficient DDoS defense with programmable switches," *IEEE/ACM Trans. Netw.*, vol. 29, pp. 1509–1526, 2021.

[26] 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," *SIGCOMM Computer Communication Review*, vol. 44, p. 87–95, 2014.

[27] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: programming platform-independent stateful openflow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 44–51, 2014.

[28] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, and T. Zhang, "On-the-fly traffic classification and control with a stateful SDN approach," in *IEEE ICC*, 2017, pp. 1–6.

[29] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *Optica JOCN*, vol. 11, pp. A84–A95, 2019.

[30] M. Monshizadeh, V. Khatri, and R. Kantola, "Detection as a service: An SDN application," in *IEEE ICACT*, 2017, pp. 285–290.

[31] D. Senie and P. Ferguson, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC 2827, May 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2827

[32] Data Plane Development Kit (DPDK*). Accessed on Apr. 2025. [Online]. Available: https://www.intel.com/content/www/us/en/developer/topic-technology/networking/dpdk.html

[33] W. Wang, X. C. Wu, P. Tammana, A. Chen, and T. S. E. Ng, "Closed-loop network performance monitoring and diagnosis with SpiderMon," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 267–285. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/wang-weitao-spidermon

[34] J. Hill, M. Aloserij, and P. Grosso, "Tracking network flows with p4," in *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, 2018, pp. 23–32.

[35] European Union, "Directive 95/46/EC. General Data Protection Regulation," Accessed on Apr. 2025. [Online]. Available: http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf

[36] California State Legislature, "California Consumer Privacy Act of 2018," Accessed on Apr. 2025. [Online]. Available: https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375

[37] Intel. (2021) Tofino native architecture – OpenTofino. Accessed on Apr. 2025. [Online]. Available: https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf

[38] Intel® Tofino™ series. Accessed on Apr. 2025. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html

[39] X. Chen, "Implementing AES encryption on programmable switches via scrambled lookup tables," in *ACM SPIN*, 2020, p. 8–14.

[40] SUP4RNET: an experimental platform running on P4 switches inside the PROGNOSE lab. Accessed on Apr. 2025. [Online]. Available: https://sup4rnet.github.io

[41] gRPC. Accessed on Apr. 2025. [Online]. Available: https://grpc.io

[42] A. Alsadi *et al.*, "No spring chicken: Quantifying the lifespan of exploits in iot malware using static and dynamic analysis," in *ASIA CCS*, 2022, p. 309–321.

[43] I. Corporation and contributors, "SDN-NFV Hands-on Samples," Accessed on Apr. 2025. [Online]. Available: https://github.com/intel/SDN-NFV-Hands-on-Samples

[44] A. Turner and contributors, "Tcpreplay: Pcap editing and replay tools for network testing," Accessed on Apr. 2025, 2000–2024. [Online]. Available: https://tcpreplay.appneta.com/

[45] H. Griffioen, G. Koursiounis, G. Smaragdakis, and C. Doerr, "Have you SYN me? Characterizing ten years of Internet scanning," in *ACM IMC*, 2024, p. 149–164.

[46] Cisco. (2024) TRex Realistic Traffic Generator.

[47] Benchmarking methodology for network interconnect devices. Accessed on Apr. 2025. [Online]. Available: https://www.ietf.org/rfc/rfc2544.txt

[48] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *SIGCOMM*, 2019, pp. 334–350.

[49] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "FCM-sketch: generic network measurements with data plane support," in *CoNEXT*, 2020, p. 78–92.

[50] V. Bruschi, S. Pontarelli, J. Tollet, D. Barach, and G. Bianchi, "FlowFight: High performance–low memory top-k spreader detection," *Comput. Netw.*, vol. 196, p. 108239, 2021.

[51] A. Cornacchia, G. Bianchi, A. Bianco, and P. Giaccone, "Staggered HLL: Near-continuous-time cardinality estimation with no overhead," *Comput. Commun.*, vol. 193, 2022.

[52] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "Beaucoup: Answering many network traffic queries, one memory update at a time," in *SIGCOMM*, 2020, pp. 226–239.

[53] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: Enabling state-intensive network functions on programmable switches," in *SIGCOMM*, 2020, p. 90–106.

[54] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "SCREAM: sketch resource allocation for software-defined measurement," in *CoNEXT*, 2015.

[55] S. Goswami, N. Kodirov, C. Mustard, I. Beschastnikh, and M. Seltzer, "Parking packet payload with P4," in *CoNEXT*, 2020, p. 274–281.

Zhihao Wang is currently working toward the Ph.D. degree in control science and engineering with the University of Electronic Science and Technology of China (UESTC). He is also a visiting Ph.D. student at Politecnico di Torino, Torino, Italy, funded by Chinese Scholarship Council (CSC). His research interests include network measurement, programmable networks and cyberspace security.

Alessandro Cornacchia is a Postdoctoral Fellow at the King Abdullah University of Science and Technology (KAUST), Saudi Arabia. He obtained his Ph.D. from Politecnico di Torino, Italy. His research interests span multiple aspects of distributed systems and data center networks, with a special appreciation for monitoring, observability and root-cause analysis. He served as a reviewer for ACM/IEEE journals and as a PC member for ACM workshops.

Andrea Bianco (Senior Member, IEEE) received the Ph.D. degree in telecommunications from the Politecnico di Torino, Italy, in 1993. He is a Full Professor and the Vice Rector of the Internal Affairs, Politecnico di Torino. He has co-authored over 230 papers published in journals and presented at leading international conferences. His research interests include design of all-optical networks, switch architectures for high-speed networks and data centers, SDN, and networked music performance.

Paolo Giaccone (Senior Member, IEEE) is a Full Professor with the Department of Electronics and Telecommunications, Politecnico di Torino, Italy. During 2000-2001 and in 2002 he was with the Information Systems Networking Lab, Electrical Engineering Dept., Stanford University. His main area of interest is the design of network algorithms, in particular for SDN networks and cloud computing systems.

Idilio Drago is an Associate Professor at the University of Turin, Italy. His research interests include network security, AI, machine learning, and Internet measurements. He is particularly interested in how AI and machine learning can help extract knowledge from network data, and secure the network. Drago has a Ph.D. from the University of Twente, the Netherlands. He was awarded the IETF/IRTF Applied Networking Research Prize.

Dingde Jiang (Member, IEEE) is currently a professor with the School of Information and Communication Engineering, UESTC. His research focuses on network measurement, modelling and optimization, network management, network security, particularly in SDN, information-centric networking, energy-efficient networks, and cognitive networks. His research is supported by the NSFC, the Program for New Century Excellent Talents with the University of Ministry of Education of China, and so on.

Marco Mellia (F'21), Ph.D., is a full professor at Politecnico di Torino, Italy. He has co-authored over 300 papers published in journals and presented at leading conferences. He won the IRTF ANR Prize at IETF-88, and the best paper awards at IEEE P2P'12, ACM CoNEXT'13, IEEE ICDCS'15, ACM CCR'16, ITC'18. He is the Editor in Chief of the Proceedings of the ACM on Networking. His research interests are in the areas of Internet monitoring, users' characterisation, cyber security, and machine learning applied to different sectors.