ChamaleoNet: Programmable Passive Probe for Enhanced Visibility on Erroneous Traffic

Zhihao Wang
University of Electronic Science and Technology of China

Alessandro Cornacchia
KAUST

Andrea Bianco
Politecnico di Torino

Idilio Drago
Università di Torino

Paolo Giaccone
Politecnico di Torino

Dingde Jiang
University of Electronic Science and Technology of China

Marco Mellia
Politecnico di Torino

Abstract

Traffic visibility remains a key component for management and security operations. Observing unsolicited and erroneous traffic, such as unanswered traffic or errors, is fundamental to detect misconfiguration, temporary failures or attacks. ChamaleoNet transforms any production network into a transparent monitor to let administrators collect unsolicited and erroneous traffic directed to hosts, whether offline or active, hosting a server or a client, protected by a firewall, or unused addresses. ChamaleoNet is programmed to ignore well-formed traffic and collect 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 SDN paradigm, ChamaleoNet processes the traffic flowing through a campus/corporate network and focuses on erroneous packets only, lowering the pressure on the collection system while respecting privacy regulations by design. ChamaleoNet enables the seamless integration with active deceptive systems like honeypots that can impersonate unused hosts/ports/services and engage with senders. The SDN in-hardware filtering reduces the traffic to the controller by 96%, resulting in a scalable solution, which we offer as open source. Simple analytics unveil internal misconfigured and infected hosts, identify temporary failures, and enhance visibility on external radiation produced by attackers looking for vulnerable services.

1 Introduction

Network monitoring has always been the first step to implement any network management or security policy. Most monitoring systems offer visibility on regular traffic to derive statistics on performance and application usage [24, 31, 53, 58, 59]; focus on protecting the network infrastructure [7, 36, 45]; or log unsolicited traffic, e.g., like network telescopes [43, 46].[1] These systems mostly ignore the *erroneous traffic*, i.e., requests to regular hosts that go unanswered, or that generate error messages at the network layer. These include the cases of external or internal clients trying to connect to offline or firewalled systems, misconfigured hosts reaching the wrong servers, routing issues, and malicious actors that look for their next targets.

In this paper, we propose ChamaleoNet, a flexible system that opportunistically permits the collection of this erroneous traffic in an operational network. ChamaleoNet logs the Internet radiation generated by *external hosts* by opportunistically observing unanswered requests destined to addresses assigned to regular hosts but that are turned off, temporary disconnected, or protected by firewalls. This mix of live and unused addresses makes attackers more engaged than in classic telescopes [49, 52].

Likewise, ChamaleoNet logs requests originated by *internal hosts*, offering visibility on misconfigured clients, infected hosts, and unveiling issues with external services.

At last, ChamaleoNet can transform itself in an active responder, enabling the personification of inactive hosts or services, further engaging with senders 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. In a nutshell, when services or hosts are not respondents, we forward their traffic to a back-end collector, either a simple passive collector or an active responder that can impersonate the original destination. Conversely, when observing a regular flow, the programmable switch filters all subsequent packets to reduce the collector load and preserve live traffic privacy.[2] 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, without affecting regular network operations. We offer

[1] A network telescope records all packets destined to an unused subnet with no connected hosts, recording the so-called Internet radiation.

[2] Here we refer to a flow as identified by the classic 5-tuple to uniquely identify a TCP or UDP flow.

Table 1: Scope of ChamaleoNet

	Traffic	Focus	Pkt	Flow	Alert	Reply	SDN
Telescope	In	Dark	✓				
Reactive Telescope	In	Dark	✓			✓	
Flow monitor	In/Out	All		✓			
IDPS	In/Out	Malicious	✓	✓	✓		
Ours	In/Out	Erroneous	✓			✓	✓

ChamaleoNet to the community as open source.[3]

We design ChamaleoNet following the data minimisation principle to respect privacy regulations and ethical principles. We collect only erroneous packets – suspicious by definition – and avoid logging all well-formed traffic. Furthermore, we collect only the data strictly required for analysis, removing any application layer payload and anonymising internal IP addresses, unless needed to generate a response. By offloading the filtering and anonymisation features to the programmable switch, ChamaleoNet enables monitoring several tens of Gbps of traffic with an off-the-shelf server. Traditional sharding and flat controller solutions can be easily adopted [35] to scale the system. We deploy and operate ChamaleoNet at our University campus network for months and present simple analytics to expose some findings ChamaleoNet enables.

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 to capture all 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 94-98%.
- ChamaleoNet improves visibility on external Internet radiation while empowering the administrators to engage with scanners by selectively enabling responders.
- Simple analytics immediately expose suspicious internal hosts, misconfigured systems, routing issues, etc., that would otherwise go unnoticed.

2 Background and Motivation

Several mechanisms exist to passively monitor traffic, from simple network telescopes to Intrusion Detection/Prevention Systems (IDS/IPS or IDPS). Telescopes collect all packets sent to unused subnets where neither servers nor clients are connected (i.e., a darknet). Post-processing the collected Internet background radiation [43, 46] gives visibility on Internet scanners, botnets, coordinated attacks, failures, routing issues and even censorship policies [9, 22, 23, 26, 33, 51]. Telescopes ignore regular and erroneous traffic and waste precious resources due to the need to leave unused large ranges of IP addresses [16]. Recent works have explored more flexible telescope designs. The work in [49] leveraged CDN infrastructure to study unsolicited traffic reaching production servers, showing how this traffic differs from traditional telescopes. Attracted by the live CDN nodes, attackers send a variety of packets that are not observed in classic telescopes. Similarly, DScope [48] places telescopes in cloud data centres. Reactive telescopes such as Spoki [30] and others [52] augment telescopes with the ability to respond to incoming requests through simple responders, 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 [7, 36, 45]. Snort, Suricata and Zeek are popular open-source solutions [55]. IDPSes react to attacks and system abuses, using known signatures or anomaly detection systems. They offer partial ability to collect packet traces, selectively logging only packets causing the incident [29] or logging all packets.

In between, network flow loggers offer visibility on operational traffic [24, 31, 53, 58, 59], with a focus on well-formed traffic for 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 1 summarises the ChamaleoNet scope. All monitoring platforms process incoming traffic, i.e., traffic initiated by an external host and directed to an internal host. Flow monitors, IDPSes and ChamaleoNet offer visibility on traffic initiated by internal hosts (outgoing traffic) too. Telescopes focus only 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 limited packet capture abilities. Only reactive telescopes can reply to incoming requests, e.g., by completing the TCP three-way handshake.

ChamaleoNet differs in several key aspects. First, it focuses specifically on erroneous traffic, mostly ignored by previous systems. Second, it offers visibility on both incoming and outgoing traffic. Third, it operates on live production networks and transparently exploits any unused addresses as a dark address, including hosts that are temporarily off-line. Fourth, it logs information at the packet level while preserving privacy through data minimisation principles. Furthermore, ChamaleoNet can selectively impersonate inactive hosts or specific services. At last, ChamaleoNet leverages the SDN programmability to scale. In a nutshell, ChamaleoNet is the only system that explicitly targets erroneous traffic that traditional systems mostly ignore. It places itself between passive telescopes and advanced flow monitors and IDPSes, offering visibility on both external and internal erroneous traffic while ignoring well-formed traffic.

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

3 CHAMALEONET design principles and architecture

We show an overview of CHAMALEONET in Fig. 1. We design it as a pluggable network system (blue box) with minimal configuration requirements. Connected to the campus network border router(s), CHAMALEONET observes all traffic entering/leaving the network and forwards only the erroneous portion to the collectors, eventually letting them impersonate the destination host or service. Observed the first packet of a new flow, it must determine whether the destination service is active, and eventually forward traffic destined to inactive services or hosts to the backend collectors. 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. We implement CHAMALEONET as a network function that takes advantage of the flexibility of the SDN paradigm. Next, we elaborate on the various components and their interactions.

3.1 The CHAMALEONET architecture

CHAMALEONET is based on an SDN-capable switch and a centralised SDN controller. We here assume an out-of-path deployment (see Sec. 7 for a discussion of the in-path alternative). All ingress and egress traffic, destined to and coming from the campus network, is mirrored to the 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.

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 (incoming and outgoing). 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* (A).

The SDN controller runs the logic to determine whether the destination of a suspicious packet (host and service) is *answered* or *unanswered*. Recall CHAMALEONET must not log any traffic destined/received to/from active services which reply to requests. For this, the controller runs a dedicated user-space *Flow-State Detection Network Function (FSD-NF)*. It 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 all following packets) are part of the legitimate flow. The controller drops the suspicious packet and installs a new rule on the SDN switch in the benign flow table (B) to drop all matching packets. The SDN switch will discard any subsequent flow packet, avoiding overflowing the controller (C).[4]
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 (D), which may log or eventually reply to such requests to engage with potential attackers acting as a honeypot.

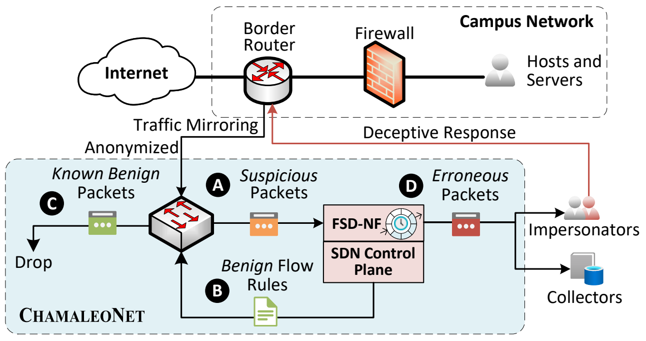


Figure 1: Overview of CHAMALEONET architecture.

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 easily 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 for the next T_{alive} period. 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.

The time diagram in Fig. 2 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.[5] Then, a response packet

[4]Due to delays and unknown sender policies, the controller shall drop any additional packets that may still be forwarded to the controller while waiting for the rule to be installed in the switch – see Sec. 6.1.

[5]Eventually, this second packet could be stored at the FSD-NF and sent to the cybersecurity collector too.

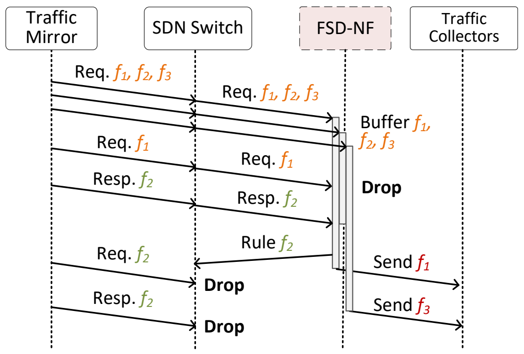


Figure 2: Expected behaviour of CHAMALEONET.

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 telescope collectors, removing any internal state related to them.

3.2 Design goals and ethics

Privacy. As the mirrored traffic includes legitimate traffic, privacy is of utter importance. We consider an "honest but curious" scenario where the person in charge of processing the data (the data processor) will not deviate from the defined goals, and will attempt to learn possible information only from legitimately received information. For this, we 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 offered by P4 [14] to perform the anonymisation step directly at the switch (see Sec. 4.3 for details).[6]

Impersonating not-responding servers or services. A fundamental requirement for CHAMALEONET is to avoid interfering with regular network activity and production traffic. Logging erroneous packets is not critical. Conversely, impersonating non-responding servers or services requires particular attention. When a regular service hosted on the campus network goes offline for maintenance or temporary failures, legitimate users could find themselves interacting with the honeypot. Likewise, impersonating an internal client when offline (e.g., a personal computer turned off at night) poses legal and security concerns if the honeypot starts interacting with malicious senders. This calls for strong safeguarding policies. For these reasons, 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 explicitly disabled for the hosts/services CHAMALEONET will impersonate so that the original IP addresses and payload are exposed to the responder.

[6]The anonymisation can also be performed on a separate device under the management of the IT network and security group if required. In this case, Cryptography-based Prefix-preserving Anonymisation [56] could be applied.

4 Implementation

Fig. 3 shows the CHAMALEONET data structures and processing. Suspicious packets arriving from the switch are managed by a *Packet Reception thread* (pkt_rx) that stores packets and handles the logic. In Sec. 4.1, we detail its implementation within the FSD-NF. A separate *Control Plane thread* (cnt_tr) manages the rule insertion and eviction at the switch, described in Sec. 4.2. Finally, we describe the implementation of the anonymiser in Sec. 4.3.

4.1 FSD-NF: a deep dive

To implement the FSD-NF, we rely on state-of-the-art per-flow monitoring based on multithreaded programming, which has been shown to cope with several tens of Gbps on off-the-shelf hardware [53]. For packet reception, CHAMALEONET hooks on `libpcap` to receive packets. We configure `libpcap` with "immediate mode", which disables packet batching and makes packets available to upper layers upon arrival to handle them to the FSD-NF. We set the capture buffer size to 2 GB to handle traffic bursts and periods in which the capture thread is stopped.[7]

Packet buffering and data structures. There are four 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; two bitmaps to keep track of which internal hosts are alive. We describe them in the following.

The FSD-NF buffers suspicious packets and waits for a response (benign flow case) or the DT timer to expire (erroneous packet case). By construction, DT timeouts for suspicious packets expire in the same order in which the packets are received. This allows us to implement a single and lazy timer and avoid the burden of managing a per-packet timer. We store packets in the FSD-NF memory, whose position is stored in descriptors that form a ring buffer queue as shown in Fig. 3. The descriptor stores also the packet arrival time *tArr* and a pointer to the entry in the flow hash table.

Benign flow detection. When the FSD-NF receives a response packet, it deems the service as active and the flow as benign. To optimise the lookup when matching responses to past suspicious packets, the hash table (with linked lists

[7]A DPDK [2] stack is compatible with the CHAMALEONET structures and transparent to FSD-NF.

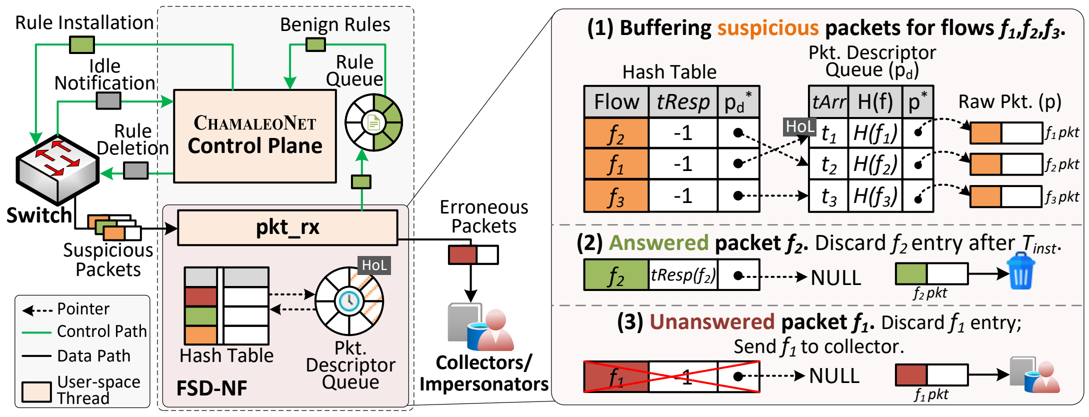


Figure 3: Implementation of ChameleoNet and its user-space packet processing workflow.

to handle collisions in the same bucket) stores a pointer to the packet descriptor position within the ring buffer. When a response packet matches a suspicious packet entry in the hash table, the FSD-NF deletes the corresponding suspicious packet from the packet descriptor and installs an entry in the switch to filter all future packets of this benign flow. At the same time, it marks the arrival time of the response *tResp* in the hash table to cope with the latency required to install the entry. During this transient phase, the FSD-NF drops any eventual other packets belonging to the benign flow.

Periodically, a cleaning routine scans the hash table and removes all the entries corresponding to benign flows whose response arrived earlier than a threshold T_{inst}. We choose T_{inst} larger than the maximum time required to install a new entry in the switch rule set. For scalability, the cleaning routine scans only a fraction $\alpha_{HT} \in (0,1]$ of the whole hash table to avoid blocking the main thread for a long time. Event (2) in Fig. 3 (right) shows the deletion of flow f_2 packet after T_{inst}.

DT timer management. 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, insert it in the queue, and set *tArr* to the current time.

When the packet descriptor queue is not empty, with periodicity P_{DT} we check the *tArr* of the oldest entry. Starting from it, we keep checking which descriptors to process by comparing the current time with its *tArr*. If the elapsed time is greater than the DT timer, we delete the hash table entry, send the packet to the collector and free the corresponding descriptor and buffer. We stop at the first descriptor that is younger than the DT timeout. To limit the computation cost of this linear search operation, we allow a maximum search depth equal to $d_{\max}$ positions.[8]

Event (3) in Fig. 3(right) illustrates this removal mechanism. In this case, when the f_1 timer expires, the packet is marked erroneous and sent to the collector.

Host liveness monitoring. To enable the controller to distinguish between erroneous packets sent to live versus dark internal destinations, ChameleoNet tracks the liveness of each internal host. A host is “alive” if we observe at least one packet from such a host – either filtered by the SDN switch in ongoing flows, or mirrored to FSD-NF in case no valid flow rule is present. To handle this, FSD-NF maintains two bitmaps to track the status of each internal host. Each bit indicates whether the associated host is live (1) or dark (0). The first bitmap records whether FSD-NF has processed any outgoing packet from the host. The second bitmap tracks whether any active flow rule matches the host in the SDN switch. This is done using the rule installation and cleaning messages described afterwards, which synchronises rule state of the SDN switch with FSD-NF. The FSD-NF merges the two bitmaps via a bitwise OR operation to define the status of the host. For each bitmap, when an entry is not refreshed for T_{alive}, its bitmap status is reset to 0.

4.2 P4 switch control plane highlights

The control plane thread cnt_tr manages flow rule installation and eviction from the P4 switch using a separate thread. Our implementation is based on the Intel Tofino switch and development kit. We use gRPC for control plane communication with the P4 switch.

Static per-service rule filter. To reduce the load on the controller, 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 $\{dstIP, port\}$. For instance, this set can include well-known and benign services frequently visited by internal clients, e.g., popular cloud or CDN services. This saves unnecessary overheads in the FSD-NF and in the control plane that should install a per-flow rule for each flow involving these known benign services. Populating this list with the top-400 most popular external services like CDNs and video servers, we whitelist up to 30%

[8]While this check could be done in a separate thread, it would create race conditions on the ring buffer and introduce synchronisation overheads. Experimental results confirm that this is detrimental and not needed (see Sec. 5).

of traffic.

Dynamic benign flow rule installation. Whenever a benign flow is detected, the pkt_rx thread pushes a rule to be installed to the control plane via a shared message-passing FIFO queue. The thread cnt_tr asynchronously pops the rules from this queue and installs them on the switch. We batch rule installation to amortise gRPC communication overheads.

We use two exact-matching rules with key *(dst IP, dst port, proto)* and *(src IP, src port, proto)* to filter all packets of a given flow independently from their direction.

All in all, the filtering mechanisms let the switch discard 95-98% of the benign traffic, with the controller that has to manage just 3% in our deployment – see App. A for details.

Idle rule cleaning. The switch stores flow rules in Match-Action Tables (MAT), whose capacity is typically limited, thus the control plane thread must evict rules by periodically deleting idle entries for (likely) terminated flows. We implement the entry eviction based on the idle notification mechanism provided by Intel Tofino architecture. Each entry in the MAT maintains a `TTL` (Time To Live) field. The switch periodically controls the rules every `Query_interval` and marks those rules for which there was no match in the previous interval as "idle". Then, it decrements the rule `TTL` by `Query_interval`. During the countdown, any new packet matching the rule resets the `TTL` to its initial value. When the `TTL` reaches zero, the switch triggers an idle timeout notification and sends it to the controller via the gRPC channel (managed in a publish-subscribe fashion). The controller receives the notifications and groups them into batches to send multiple rules deletion commands within a single gRPC call for better performance. This message is also used to reset to 0 the status of switch host aliveness bitmap. We tune the batch size in Sec. 6.

4.3 Traffic anonymizer

Privacy regulations such as GDPR [25], CCPA [17] 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.[9] 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 (TNA) [4, 32]. In such architecture, packets first traverse the ingress pipeline before being buffered at the Traffic Manager (TM) and processed at the egress pipeline. Packet processing through a sequence of MATs may happen at the ingress and egress stages pipeline, with the possibility to add some metadata to coordinate the processing.

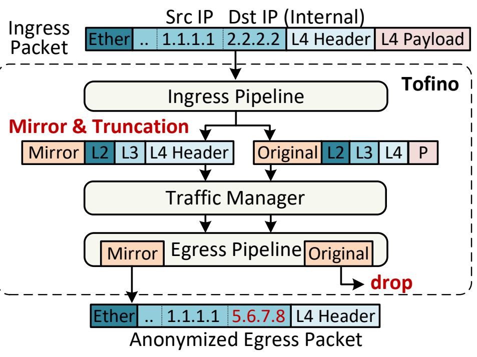


Figure 4: P4-based traffic anonymiser implemented on a Intel Tofino [4] programmable chipset.

Considering IP address obfuscation of internal host IP addresses, we rely on consistent obfuscation by xoring the original value with a predetermined key salted with the last byte of the address itself. More complex techniques can be adopted [18, 38].

Conversely, supporting L4 anonymisation through packet truncation requires some ingenuity. We leverage mirroring sessions. 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. Referring to Fig. 4, 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 the controller, and configures the output of the original packet as DROP. When the mirrored packet arrives at the traffic manager, 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. 4, 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.

5 CHAMALEONET in action

We implemented the CHAMALEONET switch functionalities using 413 lines of $P4_{16}$ [14] code, using C/C++ and Python

[9]These privacy policies have been discussed and validated by our Data Protection Officer (DPO).

Probes	ASs	IPs	Acknowledged Scanners	Ports	$\overline{IP}$	σ_{IP}	σ_{Port}
/23 telescope	715	7k	708	1k	203	11	77
CHAMALEONET	2310	17k	935	62k	203	28	210

Table 2: Quantitive comparison between traffic collected by CHAMALEONET and pure telescope every hour.

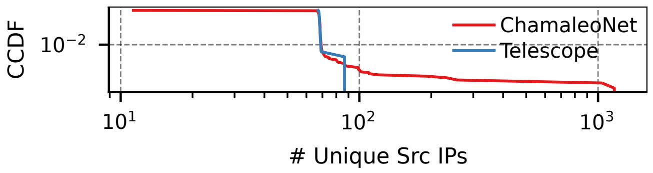


Figure 5: Unique source IPs per host distribution.

to implement the FSD-NF and controller functionalities. The controller leverages the Barefoot Runtime Interface (BRI) to program the Tofino switch and communicates via a gRPC [3] channel. We use the SUP4RNET [6] platform 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 [4]. We run the FSD-NF and CHAMALEONET controller on a virtual machine (VM) featuring 16 logic cores and 16 GB RAM.

We have deployed CHAMALEONET in our campus network for 5 months. Our network is connected to the Internet with a 20 Gbps bidirectional link. At peak time, the border router forwards to CHAMALEONET around 16 Gbps of total traffic. During this period, CHAMALEONET collects data involving 34,304 internal IPv4 addresses.[10] 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. 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 with no service attached. It has been operating for more than 7 years to observe only Internet radiation.

Below, we report some findings obtained by running simple analytics on the collected traces.

5.1 External erroneous radiation

CHAMALEONET offers visibility on external erroneous traffic, behaving as a hybrid telescope that mixes dark and live hosts. To this end, we investigate whether and how the addresses monitored by CHAMALEONET attract different traffic compared to the pure telescope. For this study, we sample 25 days of data collected from December 7 to 31, 2024.

Number of external senders. Table 2 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[11]. This is expected since CHAMALEONET observes a larger network (/17 vs /23 of the telescope). Yet, senders in CHAMALEONET are also significantly more aggressive, contacting almost the whole port range (62×).

Let us now focus on the number of external senders that contact each internal IP address. On average, each internal host receives traffic from 203 unique external senders, denoted as $\overline{IP}$ in both deployments. This is expected because the majority of hosts in the campus network function as "dark" hosts. Yet, the variance in the number of external senders per internal host, σ_{IP}, and in the number of contacted ports, σ_{port}, is much larger in CHAMALEONET. This happens because external senders get more engaged with some specific destinations observed only by CHAMALEONET. The CCDF of $\overline{IP}$ in Fig. 5 shows that the most frequently targeted hosts see one order of magnitude more unique senders. Those are the internal campus servers that expose active services.

Fig. 6 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 [51], the presence of active services causes a significant increase in unsolicited traffic, causing the per-port probability to change.

In a nutshell, CHAMALEONET collects much more Internet radiation than a pure telescope. As [48, 49] 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. 7 illustrates the erroneous packets for each adress 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-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

[10]Our campus uses one /17 subnet and 6 /24 subnets, for a total of 34,304 addresses.

[11]Ground truth information from https://gitlab.com/mcollins_at_isi/acknowledged_scanners

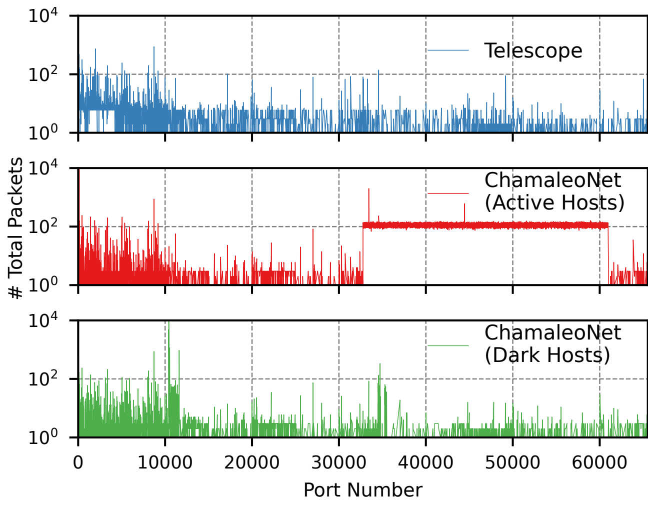


Figure 6: Erroneous packets per destination port on the telescope, active hosts and dark hosts.

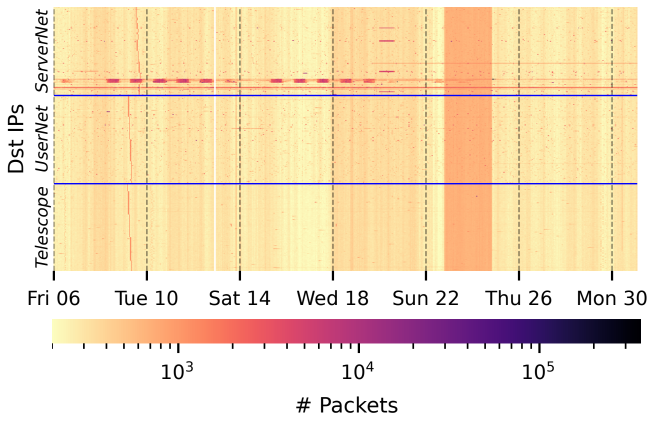


Figure 7: Comparison of incoming traffic collected by CHAMALEONET and pure telescope on a 1-hour time interval. Blue lines separate the three /24 subnets.

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 only 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), showing the temporal evolution of these external senders in Appendix B.

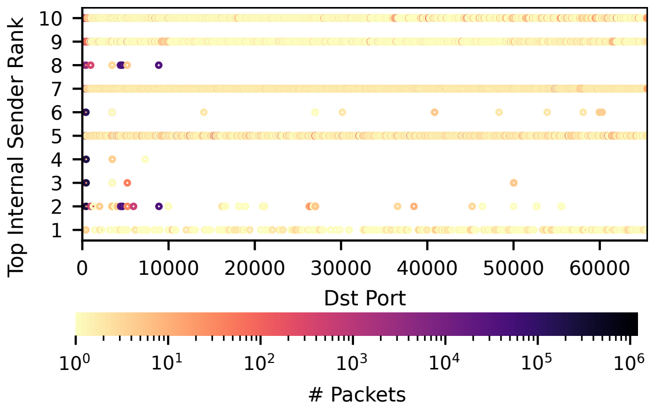


Figure 8: TCP destination ports of internal erroneous traffic generated by the top-10 contributors.

- 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: This sender scans ports 37215, 52869, and 49152 on thousands of hosts, hinting at a Satori bot [8]. It also scans other ports, such as 1900, and 2048, indicating broader scanning behaviour. The sender is present in online blocklists.
- IP 193.X.X.X: This sender 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: This sender consistently contacts port 53137 on the same internal server from source port 57498. It sends about 720 packets per hour. This indicates a possible 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 responders to engage with scanners. To understand which services the scanners are interested in, we activate impersonating TCP responders on some unused IP addresses on the campus for three hours. These responders complete the TCP three-way handshake, observe the first application message, if any, and tear down the connection. We run nDPI [24] 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 [52]. The

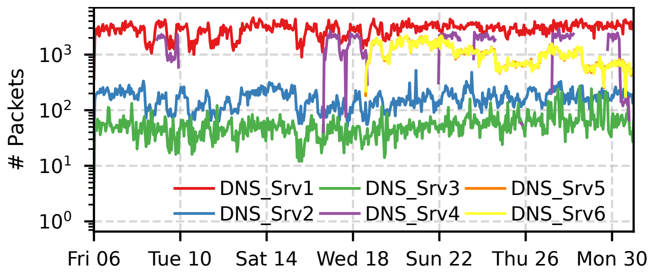


Figure 9: Erroneous DNS traffic sent by the campus DNS servers to external DNS servers.

impersonating ability of CHAMALEONET is fundamental for exploring the intention of senders.

5.2 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. 8 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 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. For completeness, we plot the temporal evolution of the radiation traffic generated by 5 of these internal hosts in Appendix C, along with some possible explanations.

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. 9. 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, likely hinting at temporary failures of some services. 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.

Understanding the precise reasons for these erroneous behaviours is outside the scope of this work. However, these simple findings illustrate the benefits of the visibility CHAMALEONET provides.

6 CHAMALEONET parameter analysis

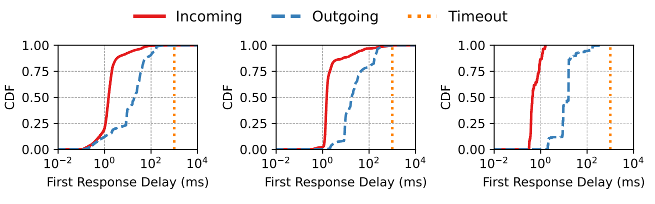
We now present results collected in controlled experiments to optimise the CHAMALEONET design.

We set up a second VM to run a traffic generator. The VM is bridged to the server E810 NIC in PCIe passthrough mode, using a separate SR-IOV [21] virtual function. We make sure packets transmitted from the traffic generator VM are forwarded to the P4 switch before coming back to the FSD-NF VM.[12] The traffic generator replays real-world traffic traces captured on our campus network. They contain about 6 million flows and 152 million packets and last 10 minutes in total. We use up to 10 `tcpreplay` [54] instances to generate different amounts of traffic and observe how CHAMALEONET behaves under different traffic loads. We split and multiplex the traces over time and replay them in parallel to mimic real-world traffic patterns. When using 10 `tcpreplay` instances we reach 7 Gbps and 1.3 Mpps, the maximum our setup allows. The goal of these experiments is to understand and optimise the internal CHAMALEONET parameters rather than stressing its scalability, which would be highly dependent on the used hardware and traffic.

6.1 Parameter setting

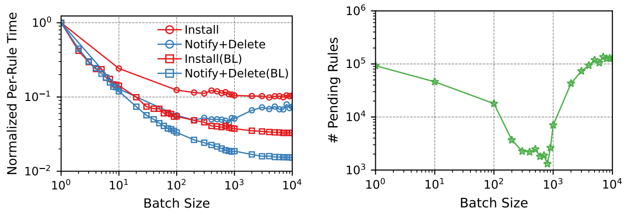
DeTection (DT) timeout. Setting the DT timeout requires some ingenuity. For this, we observe the empirical response delay statistics of actual flows. We expect it to be related to the internal and external Round Trip Time (RTT). We analyse the campus traces to measure the *first* response delay of answered flow, i.e., the time elapsed between the first request 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.

[12]We assign to the SR-IOV VFs two MAC addresses and never use those as destination MACs in the generated packets.



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

Figure 10: First response delay of incoming and outgoing flows, measured in our campus network.



(a) Rule installation/deletion. **(b) Pending rule queue.**

Figure 11: Efficiency of updating switch rules from remote BFrt clients for various batch size settings.

Fig. 10 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 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.[13] 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, e.g., using different DTs for incoming and outgoing requests, as done in telescope-like systems [28, 30, 49].

Optimising flow table updates in the switch. As described in Sec. 4, ChamaleoNet control plane runs in a separate thread and installs/deletes the rules in batches to amortise the time-consuming BFrt gRPC call. We use a FIFO queue to store pending rules generated by the pkt_rx thread while the controller processes and installs them in the switch. We preliminary measured the time to install and delete a single rule through one gRPC call, pinpointing a bottleneck.

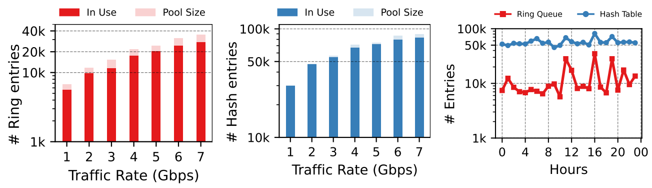
To cope with this problem, we adopt batching operations in a single gRPC call. To optimise the batch size, we randomly generate a batch of k new rules, with $k \in [1, 10^4]$. Given a batch, we install all k rules within a single gRPC call. Then, we wait for k notifications to be received from the switch before deleting all installed rules in a batch. We measure the time spent installing or deleting one rule on average, and the average queue length storing the pending rules.

We run ChamaleoNet with the traffic generator running 10 `tcpreplay` instances. Fig. 11a shows the average per-rule installation time (red) and the average per-rule deletion time (blue), normalized to the maximum measured latency. The circled lines measure at FSD-NF, so include the NF-controller interaction time and the gRPC call time. The squared lines measure at controller as pure gRPC call time and are referred as BaseLine (BL).

Results show that the batch processing policy allows us to amortise the cost of the gRPC calls very efficiently, with a speed-up factor of up to 20× when k is in the $[200 : 1000]$ range. We observe an interesting trade-off. The system improves its performance with k up to 800. Then, the additional delay it takes the switch to install a large number of rules cancels the benefit of sending all k rules with a single gRPC call. This is even worse for rule deletion because the control plane consumes time to process idle notifications, which arrive at irregular intervals. To complement this finding, we report the number of pending rules in the queue in Fig. 11b. The results confirm the need to properly choose the batch size k: if too small or too large, the system becomes unstable and the number of rules in the FIFO queue grows uncontrolled. Clearly, the smaller the batch, the more efficient the switch filtering is. We set $k = 200$, and the maximum latency T_{inst} equal to 1 s, large enough to let the switch install the new rule.

Resource utilization. As described in Sec. 4.1, we use three main data structures, i.e., the hash table, the packet descriptor queue and the packet buffer. To evaluate the resource utilisation 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. 12a and 12b show the usage of packet buffer (equal by construction to the descriptor queue) and hash table, respectively. Notably, we collect results at ChamaleoNet with all filtering mechanisms enabled. As expected, with increasing load, the memory usage grows to 20K ring entries and 80K hash table entries.

[13] In case the collector has to impersonate a not-responding host, we lower the DT timer for these hosts to avoid making the sender suspicious with delayed answers.



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

Figure 12: Resource utilization of FSD-NF for replayed traces (a)–(b) and real deployment (c).

With this, we set the hash table size equal to $20M$ buckets and $\alpha_{HT} = 10^{-3}$. For completeness, Fig. 12c 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.

Impact of synchronous periodic timeout checking. As elaborated in Sec. 4, we adopt a single thread pkt_rx both to process incoming suspicious packets and manage DT timeout expiration. Every P_D 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 $d_{\max}$. While executing this lazy strategy, we may pop several null packet descriptors corresponding to packets that were answered since the last periodic check. We skip them until a valid HoL descriptor comes at hand. 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 10 `tcpreplay` load, we configure the system to perform checks at a rate of $d_{\max}/P_D = 1$ million per second, and repeat the experiment three times to mitigate randomness. We show results in the top plot of Fig. 13 for different combinations of P_D and $d_{\max}$ and for the 75, 95 and 99 percentiles. As expected, 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 $P_D = 0.01$ ms and $d_{\max} = 10$ provides the best trade-off. 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 stored in the ring queue. Ideally, they should stay there 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. We show results in the bottom plot of Fig. 13. The configuration with $P_D = 0.1$ 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.

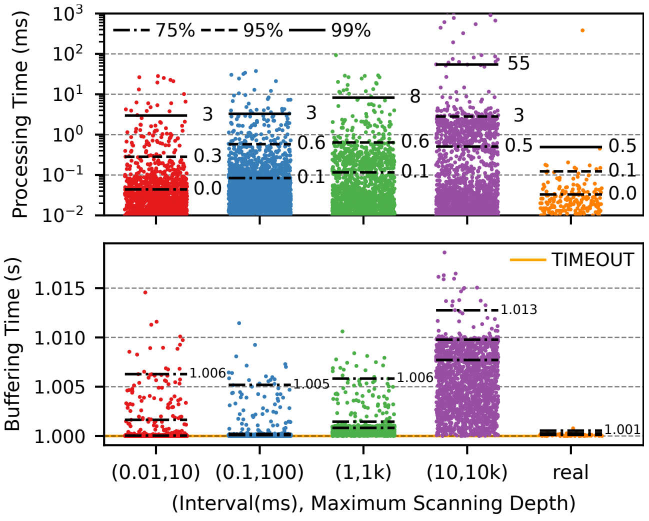


Figure 13: Periodic check of timeout expiration within pkt_rx thread for various configurations of $(P_D, d_{\max})$, with DT timeout set to $1s$. "Real" refers to the deployment in our campus network.

In general, splitting timeout checks into more frequent, smaller-scale operations improves per-packet performance and precision in managing DT timeouts. Notice that the longer DT causes only some additional memory usage without posing functional issues. Only when CHAMALEONET activates backend responders, the excessive buffering delay results in delayed replies and may limit the engagement with attackers. In this case, we reduce the DT timeout for the impersonated hosts to 70 ms.

7 Discussion

Scalability. CHAMALEONET can easily scale out by replicating the FSD-NF and the pending rule queue (Sec. 3.1) on multiple cores, e.g., with RSS [5] enabled. The control plane thread can then cycle through these queues in a round-robin fashion. We run a maximum throughput test on a single FSD-NF instance, following RFC 2544 [1] recommendations. With the current implementation, we achieve a peak throughput of around $180k$ flows/s during a synthetically injected DDoS attack.[14] DDoS attacks or horizontal scanning attacks represent worst case scenarios for CHAMALEONET, as all incoming packets correspond to a new flow, translating into a large burst of suspicious packets to be processed at FSD-NF. Notice that scaling to line rate worst case is not an imperative requirement for CHAMALEONET, which focuses on enhancing visibility and may drop excess traffic during attacks – e.g., in the switch itself [15, 19, 20, 41, 50] – once collected enough evidence.

Offloading FSD-NF to programmable dataplanes. We deploy a single FSD-NF instance and successfully operate it

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

with live traffic for months. We found the major scalability bottleneck for ChamaleoNet is not packet processing throughput at FSD-NF, but rather the limit switch Match Action Table (MAT) size and the speed in updating stored rules [57], which bounds the number of benign flow rules we can install.

Implementing FSD-NF in the dataplane diminishes the pressure on switch MATs due to benign flow rules. Indeed, if the detection of benign flows (i.e., FSD-NF) ran in the dataplane, one could directly cache the inactive services in SRAM registers, either bypassing the installation of MAT rules entirely, or balancing entries across MATs and SRAMs. There are at least two fundamental challenges in this design. First, how to access and discard suspicious packets when receiving a matching response is non-trivial, given the lack of random access memory on programmable switch ASICs. Exploring ChamaleoNet integration with state-of-the-art work in this space, such as PayloadPark [27], is left as a future direction. Second, buffering suspicious packets 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 [37,44]) would be filled up in about a millisecond at 100Gbps. Thus, despite today's programmable switch ASICs deliver Tbps throughput, ChamaleoNet cannot fully benefit from it.

In-path deployment. The proposed architecture (Sec. 3.1) 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 SDN controller shall observe all suspicious packets and install a rule to *forward* benign flow packets to the campus network instead of *dropping* them. Network administrators may be reluctant to introduce ChamaleoNet on the live traffic path. The out-of-path design illustrated in Fig. 1 offers a less intrusive and pluggable solution.

8 Related work

We complement the overview of monitoring solutions provided in Sec. 2, focusing on works that leverage SDN solutions. A recent survey summarises past works that leveraged the use of SDN as a Defence Mechanism (SaaDM) [10]. The authors provide a comprehensive taxonomy of past works based on many classification categories and approaches. The most relevant works to our are related to honeypot techniques that are deployed to detect, deflect, or stop attempts at unauthorised usage of a system. The study in [40] introduced a honeynet based on SDN, akin to ChamaleoNet, which duplicates traffic to the honey-network for further examination. In [34], authors proposed a solution where the controller inspects suspicious traffic packets and redirects them to the honeypot for additional scrutiny. Some methods integrate Moving Target Defence (MTD) with honeypots to counteract attacks. For instance, [11] proposed a virtual network architecture that utilises the controller to dynamically generate and manage flow rules to direct and control network traffic. The work in [39] targeted 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. All the previous works did not exploit the synergy between programmable switches and the transparent logic adopted in ChamaleoNet logic to redirect only a fraction of the traffic to the telescope nodes, by keeping a basic flow state machine in the FSD-NF.

In ChamaleoNet, the combination of the SDN switch and the FSD-NF enables a stateful processing of the incoming traffic. Some past works have focused on implementing a stateful processing only in the SDN switch, enabled by P4 [14] or OpenState [12]. The work in [13] proposed a reactive traffic control application that redirect traffic in real-time to a traffic classification engine. Differently from out FSD-NF, the stateful logic is implemented internally to the SDN switch. The work in [47] 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 [42] 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 attack flows. The proposed architecture is similar to ChamaleoNet, but the IDS application has a different traffic control logic.

9 Conclusions

We presented ChamaleoNet, a novel system that transforms any production network into a passive yet programmable probe to monitor erroneous traffic – such as unanswered, misrouted, or malformed packets – typically overlooked by conventional monitoring solutions. 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 five months demonstrated 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.

References

[1] Benchmarking methodology for network interconnect devices. https://www.ietf.org/rfc/rfc2544.txt, accessed on Apr. 2025.

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

[3] gRPC. https://grpc.io, accessed on Apr. 2025.

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

[5] Receive side scaling on Intel® network adapters. https://www.intel.com/content/www/us/en/support/articles/000006703/ethernet-products.html, accessed on Apr. 2025.

[6] SUP4RNET: an experimental platform running on P4 switches inside the PROGNOSE lab. https://sup4rnet.github.io, accessed on Apr. 2025.

[7] Oluwadamilare Harazeem Abdulganiyu, Taha Ait Tchakoucht, and Yakub Kayode Saheed. A systematic literature review for network intrusion detection system (IDS). *Springer International Journal of Information Security*, 2023.

[8] Arwa Abdulkarim Al Alsadi, Kaichi Sameshima, Jakob Bleier, Katsunari Yoshioka, Martina Lindorfer, Michel van Eeten, and Carlos H. Gañán. No spring chicken: Quantifying the lifespan of exploits in iot malware using static and dynamic analysis. In *ASIA CCS*, 2022.

[9] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jake Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Damian Kumar, Chad Lever, Zane Ma, Joshua Mason, David Menscher, Chris Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai Botnet. In *USENIX Security Symposium*, 2017.

[10] Believe Ayodele and Victor Buttigieg. SDN as a defence mechanism: A comprehensive survey. *Springer International Journal of Information Security*, 2024.

[11] Ilias Belalis, Georgios Kavallieratos, Vasileios Gkioulos, and Georgios Spathoulas. Enabling defensive deception by leveraging software defined networks. In *IARIA INTERNET*, 2020.

[12] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. OpenState: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 2014.

[13] Andrea Bianco, Paolo Giaccone, Seyedaidin Kelki, Nicolas Mejia Campos, Stefano Traverso, and Tianzhu Zhang. On-the-fly traffic classification and control with a stateful SDN approach. In *IEEE ICC*, 2017.

[14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014.

[15] Valerio Bruschi, Salvatore Pontarelli, Jerome Tollet, Dave Barach, and Giuseppe Bianchi. FlowFight: High performance–low memory top-k spreader detection. *Elsevier Computer Networks*, 2021.

[16] CAIDA. The UCSD Network Telescope. https://www.caida.org/projects/network_telescope/, accessed on Apr. 2025, 2024.

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

[18] Xiaoqi Chen. Implementing AES encryption on programmable switches via scrambled lookup tables. In *ACM SPIN*, 2020.

[19] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. In *ACM SIGCOMM*, 2020.

[20] Alessandro Cornacchia, Giuseppe Bianchi, Andrea Bianco, and Paolo Giaccone. Staggered HLL: Near-continuous-time cardinality estimation with no overhead. *Elsevier Computer Communications*, 2022.

[21] Intel Corporation and contributors. SDN-NFV Hands-on Samples. https://github.com/intel/SDN-NFV-Hands-on-Samples, accessed on Apr. 2025, 2024.

[22] Alberto Dainotti, Alistair King, and Kimberly Claffy. Analysis of Internet-wide probing using darknets. In *ACM BADGERS*, 2012.

[23] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. Analysis of country-wide Internet outages caused by censorship. In *ACM IMC*, 2011.

[24] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *IEEE IWCMC*, 2014.

[25] European Parliament and Council of European Union. Directive 95/46/EC. General Data Protection Regulation. http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf, accessed on Apr. 2025.

[26] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, Zied Ben Houidi, and Dario Rossi. i-DarkVec: Incremental embeddings for darknet traffic analysis. *ACM Transactions on Internet Technology*, 2023.

[27] Swati Goswami, Nodir Kodirov, Craig Mustard, Ivan Beschastnikh, and Margo Seltzer. Parking packet payload with P4. In *ACM CoNEXT*, 2020.

[28] Harm Griffioen, Georgios Koursiounis, Georgios Smaragdakis, and Christian Doerr. Have you SYN me? Characterizing ten years of Internet scanning. In *ACM IMC*, 2024.

[29] Pynbianglut Hadem, Dilip Kumar Saikia, and Soumen Moulik. An SDN-based intrusion detection system using SVM with selective logging for IP traceback. *Elsevier Computer Networks*, 2021.

[30] Raphael Hiesgen, Marcin Nawrocki, Alistair King, Alberto Dainotti, Thomas C. Schmidt, and Matthias Wählisch. Spoki: Unveiling a new wave of scanners through a reactive network telescope. In *USENIX Security*, 2022.

[31] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 2014.

[32] Intel. Tofino native architecture – OpenTofino. https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf, accessed on Apr. 2025, 2021.

[33] 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.

[34] Meatasit Karakate, Hiroshi Esaki, and Hideya Ochiai. SDNHive: A proof-of-concept SDN and honeypot system for defending against internal threats. In *ACM ICCNS*, 2022.

[35] Murat Karakus and Arjan Durresi. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks, Elsevier*, 2017.

[36] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Springer Cybersecurity*, 2019.

[37] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. TEA: Enabling state-intensive network functions on programmable switches. In *ACM SIGCOMM*, 2020.

[38] Hyojoon Kim and Arpit Gupta. ONTAS: Flexible and scalable online network traffic anonymization system. In *ACM NetAI*, 2019.

[39] Guanyu Li, Menghao Zhang, Shicheng Wang, Chang Liu, Mingwei Xu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, and Jianping Wu. Enabling performant, flexible and cost-efficient DDoS defense with programmable switches. *IEEE/ACM Transactions on Networking*, 2021.

[40] Ruidong Li, Minjiao Zheng, Donglin Bai, and Zhengduo Chen. SDN based intelligent honeynet network model design and verification. In *IEEE MLISE*, 2021.

[41] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *ACM SIGCOMM*. 2019.

[42] Mehrnoosh Monshizadeh, Vikramajeet Khatri, and Raimo Kantola. Detection as a service: An SDN application. In *IEEE ICACT*, 2017.

[43] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Network telescopes. Tech. report, https://escholarship.org/uc/item/1405b1bz, accessed on Apr. 2025.

[44] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. SCREAM: sketch resource allocation for software-defined measurement. In *ACM CoNEXT*, 2015.

[45] Merve Ozkan-Okay, Refik Samet, Ömer Aslan, and Deepti Gupta. A comprehensive systematic literature review on intrusion detection systems. *IEEE Access*, 2021.

[46] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson. Characteristics of Internet background radiation. In *IMC*, 2004.

[47] 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*, 2019.

[48] Eric Pauley, Paul Barford, and Patrick McDaniel. DScope: A cloud-native Internet telescope. In *USENIX Security*, 2023.

[49] Philipp Richter and Arthur Berger. Scanning the scanners: Sensing the internet from a massively distributed network telescope. In *ACM IMC*, 2019.

[50] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. FCM-sketch: generic network measurements with data plane support. In *ACM CoNEXT*, 2020.

[51] Francesca Soro, Idilio Drago, Martino Trevisan, Marco Mellia, João Ceron, and José J. Santanna. Are darknets all the same? On darknet visibility for security monitoring. In *IEEE LANMAN*, 2019.

[52] Francesca Soro, Thomas Favale, Danilo Giordano, Idilio Drago, Tommaso Rescio, Marco Mellia, Zied Ben Houidi, and Dario Rossi. Enlightening the darknets: Augmenting darknet visibility with active probes. *IEEE Transactions on Network and Service Management*, 2023.

[53] Martino Trevisan, Alessandro Finamore, Marco Mellia, Maurizio Munafò, and Dario Rossi. DPDKStat: 40Gbps statistical traffic analysis with off-the-shelf hardware. Tech. Report, https://nonsns.github.io/paper/DPDKStat-techrep.pdf, accessed on Apr. 2025, 2016.

[54] Aaron Turner and contributors. Tcpreplay: Pcap editing and replay tools for network testing. https://tcpreplay.appneta.com/, accessed on Apr. 2025, 2000–2024.

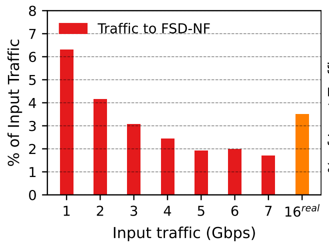
[55] Abdul Waleed, Abdul Fareed Jamali, and Ammar Masood. Which open-source IDS? snort, suricata or zeek. *Elsevier Computer Networks*, 2022.

[56] Jun Xu, Jinliang Fan, Mostafa H Ammar, and Sue B Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *IEEE ICNP*, 2002.

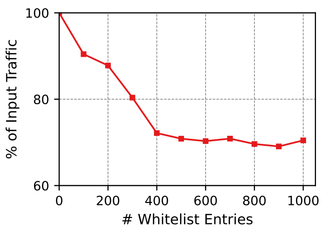
[57] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, Xiongfei Geng, Tao Feng, Feng Ning, Kai Chen, and Chuanxiong Guo. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *USENIX NSDI*, 2022.

[58] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. FlowMon-DPDK: Parsimonious per-flow software monitoring at line rate. In *IFIP TMA*, 2018.

[59] Tianzhu Zhang, Leonardo Linguaglossa, Massimo Gallo, Paolo Giaccone, and Dario Rossi. FloWatcher-DPDK: Lightweight line-rate flow-level monitoring in software. *IEEE Transactions on Network and Service Management*, 16(3):1143–1156, 2019.



(a) Under different traffic load.



(b) With different numbers of whitelist entries.

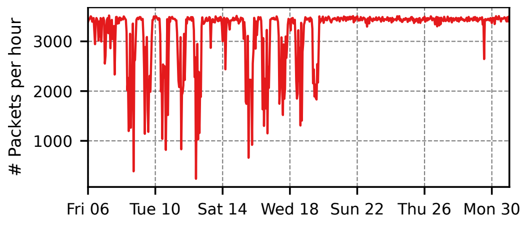
Figure 14: Filtered traffic to FSD-NF. "Real" refers to the deployment in our campus network.

A Filtering efficiency

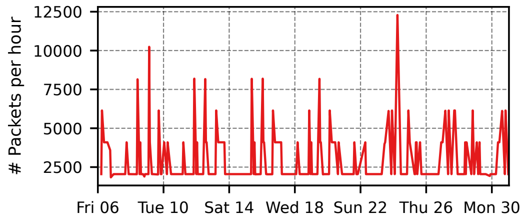
For completeness, Fig. 14 shows the efficiency of the flow filters at the switch. The left plot reports the percentage of traffic (in packets) that the controller receives under different replayed traffic loads and in real deployment. All in all, the per-flow and whitelist filters are very efficient and remove 94-96% of traffic. The service whitelist contributes up to 30% when we include the top-400 or more popular services.

B Radiation traffic by 5 selected external senders

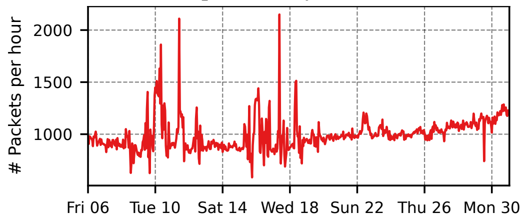
Fig. 15 shows the radiation generated by the 5 selected external senders over time. We add a brief description of the traffic they generate in the figure caption.



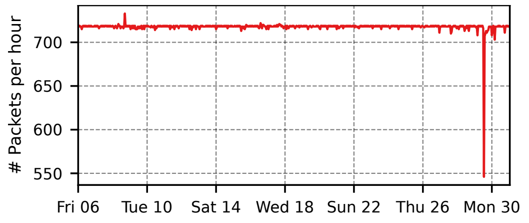
(a) 5.96.X.X - possible NTP amplification attack. All packets are UDP/123, come from the same sender and go to the same internal server.



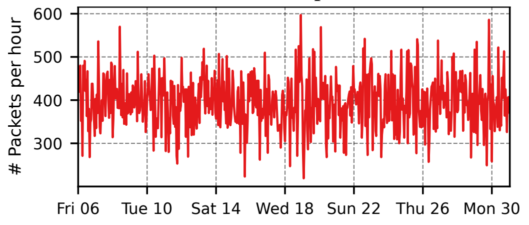
(b) 188.92.X.X - Possible Satori bot scanner. Packets go to known Satori ports and try to reach a lot of internal IP addresses (which are protected by the firewall).



(c) 193.X.X.X - ICMP Time Exceeded messages sent to more than 300 internal client. Likely a routing issue that makes the final destination unreachable for internal hosts.



(d) 94.143.X.X - possible misconfiguration or service failure. All packets come from the same sender and port, and are destined to the same host and port.

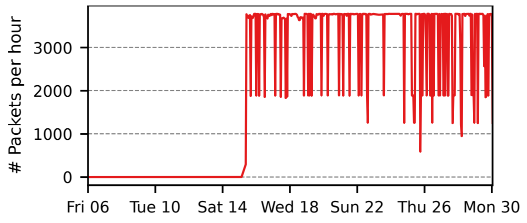


(e) 49.232.X.X - one of the 38 ping scanners. ICMP echo requests are directed to 20 internal IP addresses.

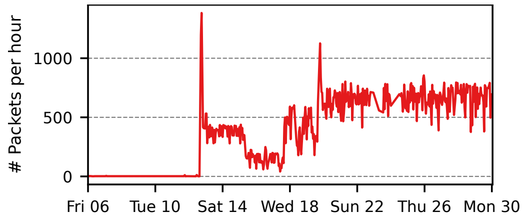
Figure 15: Temporal evolution of 5 selected campus-specific senders. Y-axis shows the average packet number for all destination (internal) IP addresses.

C Radiation traffic by top-5 internal scanners

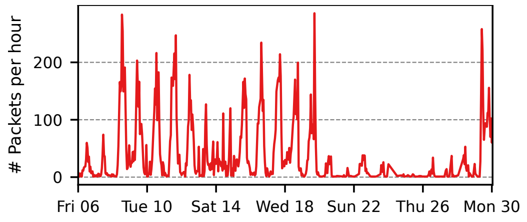
Fig. 16 shows the radiation generated by 5 example internal scanners (among top 10) over time. As before, we add a brief description in the figure caption.



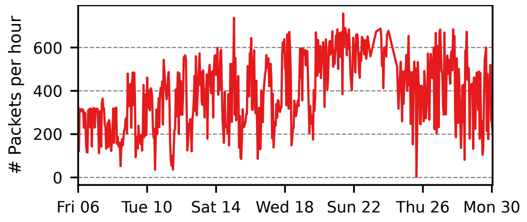
(a) 1th internal scanner - horizontal/a period/all ports - likely misconfigured repository.



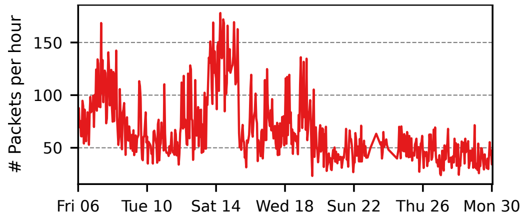
(b) 4th internal scanner - horizontal/a period/all ports - likely a scanning bot



(c) 9th internal scanner - horizontal/mostly on working hours/most ports.



(d) 3th internal scanner - vertical/all time/few ports.



(e) 7th internal scanner - vertical/all time/few ports.

Figure 16: Temporal evolution of 5 example internal scanners (among top 10). Y-axis shows the total packet number for all destination (external) IP addresses.