α-MON: Anonymized Passive Traffic Monitoring

Thomas Favale[†], Martino Trevisan[†], Idilio Drago[‡], Marco Mellia[†]
[†]Politecnico di Torino, first.last@polito.it
[‡]University of Turin, idilio.drago@unito.it

***Abstract*—Packet measurements are essential for several applications, such as cyber-security, accounting and troubleshooting. They, however, threaten privacy by exposing sensitive information. Anonymization has been the answer to this challenge, i.e., replacing sensitive information by obfuscated copies. Anonymization of packet traces, however, comes with some drawbacks. First, it reduces the value of data. Second, it requires to consider diverse protocols because information may leak from many non-encrypted fields. Third, it must be performed at high speeds directly at the monitor, to prevent private data from leaking, calling for real-time solutions.**

We present α-MON, a flexible tool for privacy-preserving packet monitoring. It replicates input packet streams to different consumers while anonymizing values according to flexible policies that cover all protocol layers. Beside classic anonymization mechanisms such as IP address obfuscation, α-MON supports α-anonymization, a novel solution to obfuscate values that can be uniquely traced back to limited sets of users. Differently from classic anonymization approaches, *α-anonymity* works on a streaming fashion, with zero delay, operating at high-speed links on a packet-by-packet basis. We evaluate α-MON performance using packet traces collected from an ISP network. Results show that it enables *α-anonymity* in real-time. α-MON is available to the community as an open-source project.

***Index Terms*—Anonymization, Passive Measurements, Traffic Monitoring, Privacy**

I. Introduction

Passive measurements collected from networks are fundamental to the well-functioning of the Internet. They are widely used to support applications such as cyber-security and traffic management. Packets flowing on network links are either saved as full-packet traces or processed on-the-fly to generate traffic summaries. Network packets however carry sensitive information about users. For example, HTTP, TLS and DNS traffic exposes names of services contacted by users, which in turn can be used to build users' profiles [1], [2]. Network measurements thus expose privacy-sensitive information and must be performed with care to avoid threatening users' privacy [3]. New privacy regulations (e.g., GDPR [4]) aim at protecting users' privacy by imposing strict rules when handling sensitive information. They provide the interested parties rights and assign powers to the regulators to enforce these rights. Network measurements must be treated in the light of these regulations, and technology must guarantee that sensitive information is not collected unless needed.

The solution to these problems has been anonymization – i.e., replacing sensitive values by obfuscated copies. Anonymization is usually done in a per-field fashion. However, different network protocol fields represent different privacy threats. Client IP addresses are *identifiers*, i.e., they allow one to immediately identify the users (devices) generating traffic. As such, they must always be obfuscated. The classic approach is CryptoPAN [5], a method that replaces IP addresses by pseudo-encrypted copies while maintaining the network prefixes. Other protocol fields, while not carring identifiers, still allow user reidentification, thus acting as *quasi-identifiers*. Server IP addresses and server names (e.g., in HTTP or TLS) are examples of quasi-identifiers. They give hints about users' interests and in some cases allow user re-identification. Quasi-identifiers therefore shall be obfuscated too.

Replacing all identifiers and quasi-identifiers in traffic measurements by obfuscated copies, however, reduces substantially the value of the traces. Taking again server names as an example, popular names (e.g., www.facebook.com or www.google.com) bring little information to uncover any specific user. Yet, being able to associate traffic to particular servers is instrumental, e.g., for network management, accounting and dimensioning.

Anonymization techniques like *k-anonymity* [6] can handle quasi-identifiers – i.e., obfuscating only values that allow user reidentification. These approaches however work with *batches* of data. They assume to have access to the complete dataset for processing and removing any combination of rare quasi-identifiers. In our scenario packets arrive at very high speeds and must be processed and forwarded online with minimum delay. Storing traces for posterior offline anonymization is not a viable option.

Here we present α-MON, a flexible and modular tool to anonymize network packets in a streaming fashion, with zero delay. α-MON acts as an anonymization device. It receives packets from the network, anonymizes them in real-time, and immediately outputs packets to multiple consumers.

α-MON follows a novel approach to anonymize packets on-the-fly. To this end, we introduce *α-anonymity*, a mechanism inspired by the *k-anonymization* principles. When observing a value in a data stream, *α-anonymization* removes it if less than α users share the value in the past ΔT time interval. Performing such fine-grained *α-anonymity* online requires ingenuity. Differently from *k-anonymity* [6], we work in a packet-by-packet basis. α-MON implements a scalable and parallelizable solution for achieving *α-anonymity*, in addition to supporting classic anonymization techniques.

We evaluate α-MON performance on Common-Off-The-Shelf (COTS) hardware with traces collected from operational networks. We show that: (i) α-MON protects sensitive data via *α-anonymity*, thus preventing the disclosure of *quasi-identifiers* uniquely associated with fewer than α users; (ii)

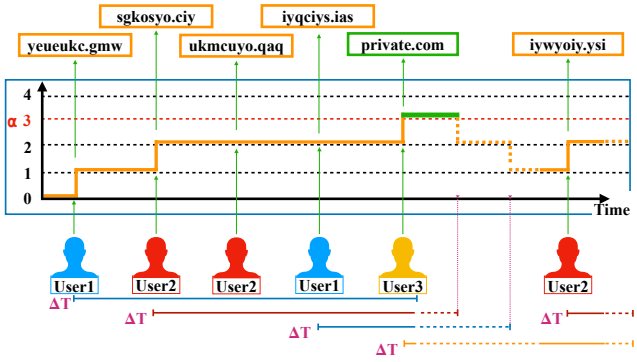


Fig. 1: α-*anonymity* concept. Three users access the domain `private.com` over time. When less than $\alpha = 3$ unique users' are seen in the past ΔT, requests must be anonymized.

α-MON allows most information that would be obfuscated by a strict per-field anonymization to be exported, thus generating richer traces than alternatives; (iii) α-MON scales to tens of Gbit/s with zero packet loss using few cores. In pessimistic scenarios, it easily achieves several Gbit/s too. Finally, α-MON is publicly available as an open-source project[1].

The paper is organized as follows. Section II defines α-*anonymity* and shows its impact on traffic measurements. Section III describes α-MON architecture, design and implementation. Section IV benchmarks α-MON performance and shows how to tune its parameters. Section V and Section VI discuss the α-*anonymity* approach and α-MON in the perspective of related work. Finally, Section VII concludes the paper.

II. α-ANONYMITY

The drastic increase in the rate at which personal data are collected pushed researchers to propose techniques to anonymize data. The goal of anonymization is to avoid disclosing personal information without compromising the utility of datasets. The seminal work of Samarati *et al.* proposes the *k-anonymity* property [6], [7]. It aims at preventing the reidentification of individuals or the extraction of sensitive information about them by ensuring that at least *k* individuals share the same properties in the dataset. k-anonimity has been extended with the *l-diversity* [8] and *t-closeness* [9] ideas, which we will discuss in Section V.

With α-MON, we want to obtain a similar effect, avoiding the exposure of sensitive information in network measurements without sacrificing their usefulness. To this end, we here introduce α-*anonymity*.

A. α-anonymity *definitions*

Standard approaches to achieve *k-anonymity* are not applicable to traffic measurements as they require the availability of the entire dataset at the anonymization time. We lie in a scenario where anonymization must be performed in real-time with zero delay and must scale up to multi-Gbit/s streams. As such, we cannot assume to have the whole dataset at disposal for anonymization.

Here, we propose a novel concept of anonymity for traffic measurements. We call it α-*anonymity*. It targets real-time, online processing, with minimum latency. *Data-subjects* (i.e., users) are identified by an *identifier*. The most common identifier in network traces are client IP addresses[2].

Quasi-identifiers are attributes whose values must be controlled, as they may help to reidentify data-subjects. In our case, quasi-identifiers are fields present in protocol headers and payload that may be associated with a small group of data subjects. Examples include server IP addresses, server names present in payloads (e.g., in DNS) and user-agent strings (e.g., in HTTP requests or QUIC handshakes). α-*anonymity* obfuscates rare values of quasi-identifiers, preventing attacks against data-subjects' privacy. We introduce the definition of α-*private quasi-identifier*.

Definition 1. An α-private quasi-identifier is a value observed at time t, which is associated with less than α data-subjects in the past ΔT time interval.

If the anonymized dataset hides α-private quasi-identifiers, it achieves α-*anonymity*.

Definition 2. A stream of packets is $\alpha - anonymized$ if all α-private quasi-identifiers are obfuscated, given α and ΔT.

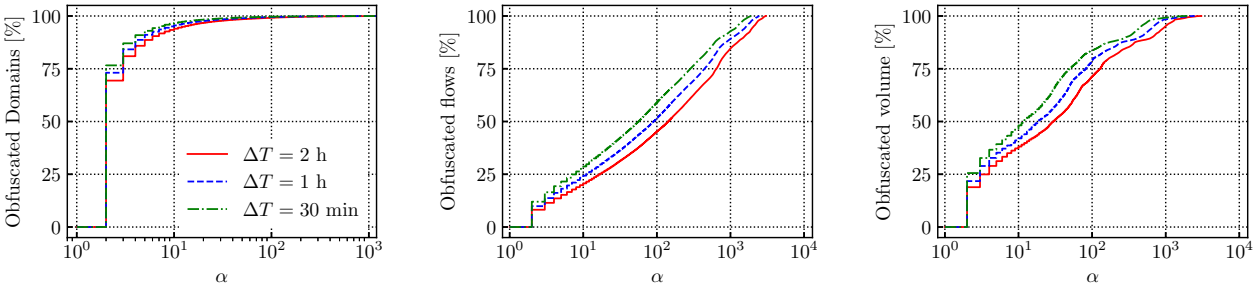
In other words, if a quasi-identifier has been observed by at most α-*1* data-subjects in ΔT, we obfuscate it. By adjusting parameters α and ΔT, it is possible to regulate the trade-off between data utility and privacy. Indeed, a large α results in the majority of values to be anonymized, while a small α allows rare values to be exposed. ΔT regulates the memory of the system.

We exemplify the idea of α-*anonymity* in Figure 1. Here the quasi-identifiers are the domain names found in packet payloads. Suppose different users access the website `private.com`. Let $\alpha = 3$. The first four accesses shall be obfuscated as only two users accessed the website up to then. When we observe User3's request, we have 3 users that have accessed `private.com` in the current ΔT. Thus, we allow User3's request to pass without anonymization. Notice that, in this case, exposing `private.com` does not uncover User3, as attackers cannot even know who the other two users are. After some time User2 accesses the domain again. The previous entry for User1 is no longer in the current ΔT window, and `private.com` is anonymized again.

Clearly, in the above example, popular websites and services would be accessed by several users. Their names would not be anonymized. Rare domain names that could bring specific information about users would likely be anonymized. Next, we show an example to illustrate the impact of α-*anonymity* on the quality of exported data.

[1] https://smartdata.polito.it/alpha-mon-anonymized-passive-traffic-monitoring/

[2] α-*anonymity* can handle any protocol field as an identifier.



(a) Domains. (b) Flows. (c) Traffic volume.

Fig. 2: Fraction of traffic obfuscated by α-*anonymity* with different values of α and ΔT.

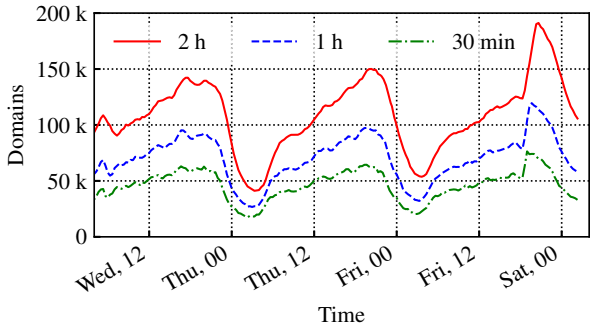


Fig. 3: Domains known by α-*anonymity* with different ΔT.

B. The case of network packet traces

To exemplify the effect of α-*anonymity* on real packet traces we consider a case where the Fully-Qualified Domain Names or *domains* in brief present in TLS, DNS, and HTTP are our quasi-identifiers. Client IP addresses are used as identifiers. We consider a traffic trace collected on an operational network including more than 8 000 users who generate several millions of packets per second of traffic.

We analyze the fraction of traffic that α-*anonymity* would obfuscate when considering different values for α and ΔT. We show results in Figure 2. First, consider the fraction of domains that α-*anonymity* would obfuscate in Figure 2a. We notice that $\alpha = 2$ already causes $\approx 75\%$ of the domains to be obfuscated. When $\alpha = 10$, the fraction increases to 90%. ΔT has a small overall impact.

Different is the picture if we consider the number of flows (Figure 2b) and the byte-wise volume (Figure 2c) carried by flows for which the domains gets obfuscated. With $\alpha = 2$, α-*anonymity* obfuscates the domains in only 10% of flows, which account for $\approx$ 25 % of the traffic volume.[3] This is caused by the nature of Internet traffic, where the majority of flows are directed towards a limited set of services [10]. The impact of a large ΔT is more pronounced for high values of α, allowing a larger number of flows to avoid obfuscation. For example, if we set $\alpha = 100$, a $\Delta T = 30min$ results in 60% of flows to be deprived of their domains; this fraction decreases to 52 (46)% if we set $\Delta T = 1h$ ($2h$). In a nut shell, popular domains that carry little sensitive information are responsible for the majority of traffic. Letting their name in clear poses no challenges for privacy, while it offers great visibility to network monitors.

An important question for practically implementing α-*anonymity* is the number of values for quasi-identifiers that α-*anonymity* has to track over time. This is fundamental to quantify its memory footprint and correctly size internal data structures (see Section III-D) as well as the ΔT parameter. For each quasi-identifier, indeed, we need to track the number of users associated with each possible value for the given quasi-identifier in the last ΔT window.

Taking again domains as an example, we consider the size of the domain set that α-*anonymity* must track – i.e., those active domains observed in the last ΔT interval. Figure 3 depicts results over time for our trace, considering three possible values for ΔT. After a short warm-up phase (not visible at this scale), the curves follow the daily trend of network usage. We observe a peak during evenings, when $\approx$ 150 000 unique domains are seen in a two-hour interval (solid red line). No more than 100 000 (60 000) unique domains appear with a ΔT of 1 hour (30 minutes). During the night, when traffic reduces, the number of active domains is more than halved. We observe a sudden peak on the evening of the third day (a Friday) with almost 200 000 unique domains accessed in two hours.

Recall that the above experiments refer to a traffic trace from a population of 8 000 users. However, given the nature of Internet traffic, where most flows are directed to few services, the set of accessed domain names scales *sublinearly* with the number of users. For example, during the peak hour, 1 000 (3 000) randomly selected users already contact 35 000 (60 000) domains, while all 8 000 users $\approx$ 150 000.

[3]Most of the obfuscated domains are names used by CDNs that include hashes or random strings as top-level domain names. Taking instead the second-level domains (`example.com` instead of `www.example.com`) as quasi-identifiers would reduce the percentage of obfuscated bytes to negligible numbers for $\alpha = 2$.

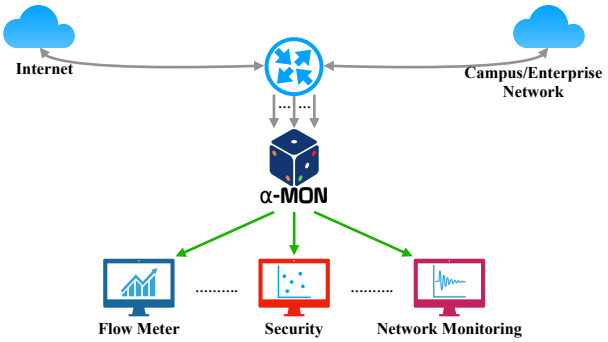


Fig. 4: Deployment scenario: α-MON anonymizes the traffic coming from a span port or optical splitter and forwards it to different legacy monitors.

III. α-MON DESIGN

We now describe α-MON, covering requirements, design choices, and implementation, with a special focus on the data structures used to perform α-*anonymity* at high-speeds.

A. Deployment scenario and requirements

Figure 4 shows the deployment scenario. α-MON operates as a classic network monitor, receiving packets from one or multiple network links, either by means of span ports or optical splitters. To allow legacy applications to coexist, α-MON is deployed in front of them and forward anonymized packets to multiple consumers. Compatible with best-practices for privacy, α-MON performs different anonymization according to the consumer, thus passing on the minimal information required by each legacy application.

α-MON must be flexible and support a rich set of functionalities. It shall satisfy the following requirements:

1) It must support α-*anonymity* to hide private quasi-identifiers with custom α and ΔT;
2) It must support a flexible set of anonymization policies, covering all protocol layers;
3) It must be scalable and deployable in high-speed links, handling multiple tens of Gbit/s with no packet loss;
4) It must support multiple legacy applications with different anonymization requirements.

B. Packet ingestion and forwarding design

α-MON runs on a COTS server and receives packets from several network interfaces. For efficiency, we implement it in C language. For packet capture, we rely on the Data Plane Development Kit (DPDK) [11], a set of libraries and drivers for fast packet processing. α-MON follows a multi-threaded design and can take advantage of all cores available in a server. We use the architecture proposed by the authors of [12], in which the incoming packets are load-balanced to different threads – one per CPU core – using the Receive Side Scaling (RSS) feature of modern network cards. Each network interface implements load-balancing algorithms so that incoming packets are spread to multiple queues based on hash functions. This mechanism allows fast and scalable load balancing in hardware and avoids wasting CPU resources.

Some of the α-MON anonymization capabilities require stateful per-flow processing and mandate data structures to keep track of the status of TCP and UDP flows.[4] To avoid expensive synchronizations, network interfaces load-balance packets in a consistent per-flow fashion. In other words, packets belonging to the same flow are always processed by the same thread. We reach this goal instrumenting the network interface with a specific RSS hash seed [13].

Each thread receives a fraction of the overall traffic. According to custom-defined configurations, packets are replicated, their payloads are anonymized and, finally, they are forwarded to output interface(s) connected to the legacy monitors. To avoid concurrent access to network interfaces, α-MON sets up a transmitting queue dedicated to each thread on each network interface, again using the DPDK functionalities.

C. Anonymization modules

We design α-MON to be modular and flexible. As such, the anonymization functions are functions that build a processing pipeline. This approach eases the configuration of anonymization policies and allows new modules to be integrated into the system with little effort. α-MON supports multiple configurations, which differ, e.g., for encryption keys and anonymization pipeline. α-MON takes care of making copies of packets and performs the desired steps on each pipeline before forwarding packets to a consumer. This design allows deployments in which different consumers require different anonymization policies, e.g., security monitors that receive original packets, while passive monitors that receive fully-anonymized packets.

Currently, α-MON implements the following modules to search and anonymize identifiers and quasi-identifiers contained in the traffic:

Layers 5-7: The key novelty of α-MON resides in the mechanisms for handling quasi-identifiers in application-layer protocols. α-MON implements a classification engine based on Deep Packet Inspection to identify popular protocols. In its current implementation, α-MON supports quasi-identifiers contained in TLS, DNS, and HTTP protocols. In particular, α-MON handles server domain names. However, any field of these protocol headers could be subjected to α-*anonymity*, with customized α and ΔT parameters. Alternatively, the fields can be obfuscated by default (i.e., treated as an identifier).

Layer 4: α-MON keeps a table to track TCP and UDP flows, allowing per-flow anonymization policies. Tracking flows is fundamental for consistent layer 5-7 anonymization. α-MON currently does not modify L4-headers, but one could easily implement a mechanism for obfuscating potentially sensitive L4 information (e.g., rarely used TCP options).

[4]We define a flow by the usual 5-tuple.

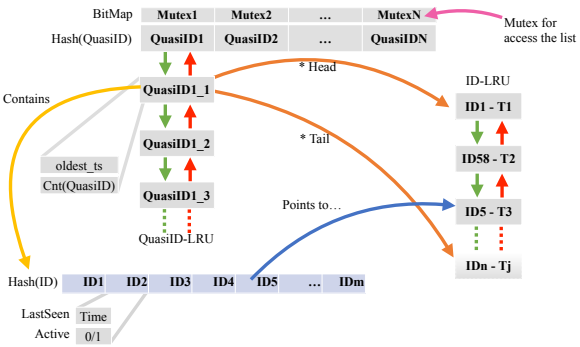


Fig. 5: Data structure used to handle quasi-identifiers.

Layer 3: α-MON considers client IP addresses as identifiers and anonymize them using the CryptoPan algorithm [14], [5]. CryptoPan encryption keys can be static or randomly rotated at fixed time intervals. α-MON allows the administrator to restrict the addresses that undergo anonymization to specific subnets, e.g., targeting only IP addresses of clients in the administered network. It supports IPv4 and IPv6.

IP addresses that are not identifiers (e.g., server IP addresses) can be treated as quasi-identifiers and undergo to α-*anonymity* too.

Layer 2: α-MON supports the removal of MAC addresses. Alternatively, as MAC addresses are generally modified by routers once they forward the packets, α-MON can store a timestamp in place of the MAC headers. This mechanism allows consumers to get timestamps of the moment packets entered α-MON, thus increasing the precision.

Finally, α-MON implements a default policy to completely drop the payload of specific/unknown protocols at any layer – e.g., forwarding only anonymized L3 or L4 headers to consumers, while removing L5-7 payloads.

D. α-anonymity *implementation and data structures*

We now describe the data structures used to implement traffic anonymization at tens of Gbit/s. To reach high speeds, it is necessary to carefully design suitable data structures that avoid expensive global synchronizations. We focus on the most challenging data structures.

α-MON includes a dedicated module for α-*anonymity*. When processing a packet from a data subject identified by ID and containing a quasi-identifier, the value of the quasi-identifier ($QuasiID$) must be evaluated. α-MON must decide whether to keep $QuasiID$ or hide it. The decision is based on the counter *Cnt(QuasiID)* of data subjects sharing the $QuasiID$ in the time window ΔT.

To keep track of these counters, we rely on the specifically designed data structure depicted by Figure 5. The data structure must be shared between all threads. Therefore, α-MON needs to handle concurrent accesses, which is a potentially expensive operation. Its core is composed of a shared hash table *Hash(QuasiID)*, in which each bucket is protected by a Mutex lock to handle concurrent accesses. A list handles hash collisions, organized as a Least Recently Used (LRU) structure for efficiency – *QuasiID-LRU* in the figure. Each entry in the LRU contains the information related to a quasi-identifier value ($QuasiID$). Beside metadata, it contains a second LRU, the *ID-LRU* list, that stores the ordered set of data subjects sharing the $QuasiID$, along with the timestamp of respective last occurrence. This *ID-LRU* is instrumental to remove those ID whose occurrences happened more than ΔT time ago.

The metadata for $QuasiID$ contains pointers to both head and tail of the ID-LRU (illustrated by orange arrows), the oldest timestamp at which $QuasiID$ has been observed and the counter of unique IDs currently active. A second inner hash table guarantees $O(1)$ access to ID-LRU elements (illustrated by blue arrows) using the ID as key in *Hash(ID)*.

When a α-MON thread has to decide whether to anonymize or not the quasi-identifier value $QuasiID$, it first accesses the hash table *Hash(Quasi-ID)*. If $QuasiID$ is empty, the corresponding entry is created; otherwise α-MON looks for $QuasiID$ through the collision list. Once found (or newly created), α-MON updates the QuasiID-LRU of the collision list, moving the current item to the top. Next, it updates the corresponding metadata for the $QuasiID$. Specifically, α-MON checks if the data subject ID is already listed among those that share $QuasiID$ in the past ΔT window. If such ID is present, its timestamp is updated to the current time. If not, the new ID is added to the ID-LRU, and the counter *Cnt(QuasiID)* of data subjects sharing $QuasiID$ is increased.

Next, we need to purge subjects that shared $QuasiID$ more than ΔT time ago. α-MON goes through the bottom of the ID-LRU backwards checking the timestamp for each entry: if the time difference with the current time exceeds ΔT, the entry is evicted. The *Cnt(QuasiID)* is decreased consequently. Similarly, α-MON periodically scans all $QuasiID$s in the hash bucket to delete those that are no longer in use.

At last, α-MON decides whether to anonymize $QuasiID$ based on the counter of the number of active data subjects. If it is smaller than α, α-MON replaces the quasi-identifier value with random bytes.

Note that α-MON is able to perform most operations in $O(1)$ for each packet, thanks to the two hash tables used to access quasi-identifier values and per-identifier counters. This design allows high processing speeds as we will show in Section IV-B.

E. Auxiliary data structures

α-MON implements efficient structures to support per-flow management. This is instrumental to apply consistent anonymization decisions based on flow state – e.g., to remove the payload of specific protocols (e.g., HTTP) or to parse application layer protocols whose fields are split across multiple packets. The data structure for active flows follows the same ideas used by the authors of [15]. It builds on a hash-based data structure that provides $O(1)$ accesses to the per-flow metadata.

IV. Experimental Results

We now evaluate the performance of α-MON. We aim at evaluating how α-MON performance scales with the number

	Flows (M)		Flows per-class (%)				Pktsize
Trace	TCP	UDP	HTTP	HTTPS	P2P	oth	avg
ISP-FULL	3.08	7.76	10.8	8.2	46.2	34.7	716
ISP-HDR	3.08	7.76	-	-	-	-	54
DNS	-	14.07	-	-	-	100	172

TABLE I: Packet traces.

of cores and the impact of different conditions and workloads. We follow the procedures defined in [16] for throughput tests.

A. Testbed and dataset

We instrument a simple testbed composed of a Traffic Generator (TG) and a Device Under Test (DUT). TG and DUT are each equipped with two quad-port Intel X710 10 Gbit/s network cards, which are directly connected. TG replays traffic traces stored in `pcap` format, sending packets to DUT over a first set of 10 Gbit/s links. The DUT runs α-MON to anonymize network traffic that is sent back to the TG over a second set of 10 Gbit/s links.

DUT is a high-end server equipped with 4 Intel Xeon Gold 6140M processors and 512 GB of memory. The total number of physical cores is 72. We disabled hyperthreading to isolate α-MON performance when varying the number of cores.

The TG is a medium-sized server with no particular requirement except for a large amount of memory. Indeed, it is not trivial to read and send stored traffic traces at tens of Gbit/s with commercial solid-state drives whose read speed is in the order of 4-5 Gbit/s. As such, we equipped the TG with 1 TB of RAM so that it can fit large traces in memory. We use DPDK-Replay to replay the traces on the selected network interfaces at the desired rate.[5] DPDK-Replay is able to loop over traces in memory, eventually replacing IP addresses on each pass, so to allow arbitrary benchmark duration.

We perform experiments using real traffic traces collected from an operational network (see Table I). Packets are captured by instrumenting a Point-of-Presence of a European Internet Service Provider (ISP) that aggregates the traffic of about 8 000 households. We capture raw packets using a passive probe equipped with several high-end SSD disks.

For the first benchmarks, we use a 1-hour long trace captured at peak time. We obtain a 575 GB of packets that we call ISP-FULL. It contains 3.1 M TCP and 7.7 M UDP flows, with an average packet size of 716 B, for more than 800 M packets. This trace represents the typical workload that α-MON would face in an ISP network.

We process this trace to keep only up to TCP/UDP headers, removing payloads. This step results in a second packet trace – called ISP-HDR– in which packets are truncated to 54 B on average. We use this trace to benchmark the per-packet capture, processing and transmission speed of α-MON in a pessimist scenario composed by lots of small packets.

At last, we collect DNS traffic from the same network for one day. We use this trace to benchmark the *α-anonymity* module since each packet likely contains a quasi-ID, e.g., a domain name. This trace is 7.23 GB large, with more than 1 M packets. We call it DNS trace.

[5]https://github.com/marty90/DPDK-Replay

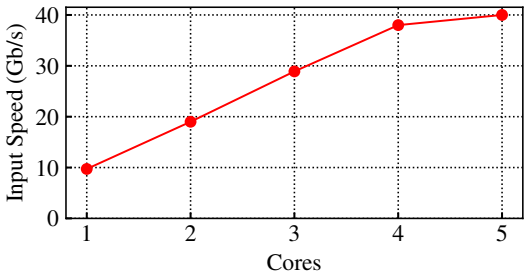


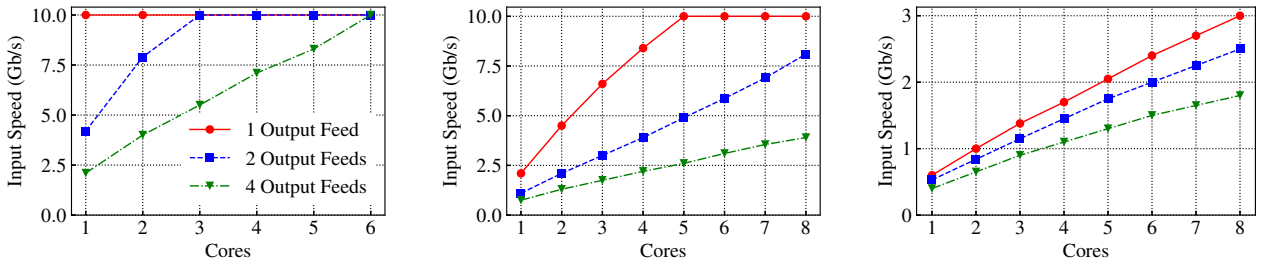
Fig. 6: Performance with four input and four output interfaces using the ISP-FULL trace.

For each experiment, we seek the *throughput* [16], i.e., the fastest rate at which the count of frames transmitted by the DUT is equal to the number of frames sent by the TG. We progressively increase the TG sending rate using a binary search process. As we increase speeds, benchmarks require the TG to perform multiple passes on the original traces. All experiments are performed with $\alpha = 10$ and $\Delta T = 60s$. Each benchmark lasts 3 minutes. We set the hash table *Hash(QuasiID)* size to 100 000 entries to maintain collision lists reasonably short (cfr. Figure 3).

B. Horizontal scalability

We first focus on α-MON horizontal scalability to understand how its throughput increases with the number of cores. Recall that each core manages an α-MON thread via DPDK. TG sends traffic to DUT using four 10 Gbit/s links. The DUT must anonymize packets before forwarding them on four output links. For each input interface, we configure one output feed on a dedicated output interface, thus, avoiding duplicating packets. Thanks to RSS load balancing, each of the N cores processes an average $1/N$ of the traffic from each input interface – $4/N$ in total, given we use 4 input interfaces. This load-balancing scheme makes the throughput depend only on the aggregate incoming rate, regardless of the rates of single input interfaces. We employ the ISP-FULL trace for this experiment.

We report results in Figure 6, which shows the throughput versus numbers of cores. When α-MON runs on a single core, it handles around 10 Gbit/s. In our experiments, the throughput is equivalent if packets come from a single input link at line rate or if they are spread on the four interfaces. With two cores, α-MON sustains 18 Gbit/s, and the performance scales linearly with additional cores, reaching 38 Gbit/s on four cores. With just five cores α-MON fully sustains 40 Gbit/s – i.e., all input interfaces at line rate. Unfortunately, our testbed does not allow higher rates due to the limited number of network interfaces, but we expect the performance to further increase



(a) ISP-FULL (b) ISP-HDR (c) DNS

Fig. 7: Performance with different traces and consumer numbers.

before hitting the PCI bus bandwidth limit [17]. We leave the study of this scenario for future work.

In summary, α-MON sustains $\approx$ 10 Gbit/s per-core on a realistic traffic trace. Its performance scales to up 40 Gbit/s when using just five CPU cores, reaching line rate on all four input links.

C. Benchmark with other workloads

We next evaluate α-MON performance under different workloads. We vary both the input traffic mixture and the number of consumers. In these experiments, the TG sends packets to the DUT using a single 10 Gbit/s link. We configure α-MON with one, two, or four output feeds, each of them anonymized using all available modules, but with different encryption keys. As such, α-MON not only has to make packet copies, but also performs all anonymization steps multiple times, thus simulating a case in which each consumer receives differently-customized traffic.

Recall that different traffic classes trigger different α-MON modules, resulting in performance variations. While the ISP-FULL trace is a *typical* workload that α-MON could face at an edge network, DNS represents an extreme scenario in which every packet triggers the α-*anonymity* module. ISP-HDR is a second extreme scenario since all packets are small. It should not be observed in practice except for anomalous situations, e.g., during cyber attacks. ISP-HDR stresses α-MON packet replication capability toward multiple consumers as well as L2-L4 anonymization modules.

We show results in Figure 7. We report throughput for different traffic traces in separate figures, where different lines indicate the number of output feeds. X-axes show the number of cores.

Figure 7a shows performance with the ISP-FULL trace. As already observed in the previous section, a single core sustains 10 Gbit/s with a single consumer (solid red line). The performance is reduced when α-MON has to feed multiple consumers. For a single CPU core (leftmost points), the throughput is reduced to 4 Gbit/s with two consumers (dashed blue line) and 2.4 Gbit/s with four consumers (dashed green line). The extra load imposed by the need for duplicating packets causes this degradation. Recall that DPDK allows zero-copy processing only when single output is required. Here, α-MON needs 3 cores to feed 2 consumers with 10 Gbit/s each, and 6 cores to feed 4 consumers. Note also how the throughput scales linearly with the number of cores in all cases. Again, contention on the *Hasd(QuasiID)* has little impact.

Next, we use the ISP-HDR trace to stress α-MON packet copying, processing and forwarding. Whereas the TG sends out 1.7 million packets per second (Mpps) when replaying the ISP-FULL trace at 10 Gbit/s, ISP-HDR results in 23 Mpps. α-MON throughput naturally decreases. A single core handles no more than 2 Gbit/s in this scenario (Figure 7b - red curve). However, thanks to the scalable architecture based on RSS, α-MON throughput increases linearly with the number of cores – and 5 cores handle 10 Gbit/s when outputting traffic to a single consumer (red line). Similar to the previous scenario, the throughput is reduced when having multiple consumers (blue and green lines). A single core can sustain 1 (0.7) Gbit/s of the ISP-HDR trace with 2 (4) consumers. Yet, throughput continues to grow linearly with the number of cores. As such, a proper resource provisioning would allow α-MON to perform its tasks without loss also in these scenarios.

Next, we consider the DNS trace to stress the α-*anonymity* module. In Figure 7c we see that throughput further decreases. Remind that packets undergoing α-*anonymity* generate updates on various data structures to track the set of data subjects associated with each quasi-identifier value. Figure 7c shows that a single core sustains 0.6 Gbit/s with one output feed. Again, the throughput increases almost linearly with the number of cores, and eight cores can handle 3 Gbit/s of pure DNS traffic. Here too, α-MON incurs a penalty for the packet copying in case of multiple consumers. The slightly sublinear scalability is due to the Mutex locking on the *Hash(QuasiID)* which slows down processing when a large number of cores are used.

In summary, α-MON can process 10 Gbit/s of typical ISP traffic with one core. Additional output feeds bring extra costs due to packet copying. A handful of cores allows achieving line rate in different scenarios. Worst-case scenarios, such as pure DNS traffic and millions of packets without payload, require a proper dimensioning of the system. Still, α-MON

performance scales linearly with the number of cores in all scenarios.

V. Discussion on the α-anonymity approach

α-anonymity represents a new proposal for anonymizing the sensitive information contained in network traffic. It shares with *k-anonymity*, *l-diversity* and *t-closeness* the idea that uncommon quasi-identifiers must be hidden to prevent users' re-identification. No scheme can provide a guarantee of anonymity, and all scheme trade privacy with utility [18]. Indeed, publishing any data results in a potential privacy loss for individuals, and any anonymization technique makes data imprecise, causing losses in potential utility. At last, efficient algorithms that provide anonymized data with such properties [19] are not well-fit for real-time and online usage as they make decisions based on the *global distribution* of quasi-identifiers. Like alternative approaches, *α-anonymity* does not offer guarantees of zero privacy loss, but allows tuning the desired trade-off between privacy and data utility.

With *α-anonymity*, we propose a novel anonymization property that can be achieved in real-time and in an online fashion. As such it is well-suited for network traffic anonymization. Unfortunately, *k-anonymity* and similar approaches work on tabular data where the entire database (or a batch of data) of entries are readily available. We instead want to anonymize a continuous stream of data and output the results in real-time. Notice that this differs from *k-anonymity* over data streams [20] - i.e., a system capable of applying *k-anonymity* on a stream database, where windows of data are considered. Such an approach is not applicable to our context, since we do not want to buffer data in windows, but we want to make the decision on a per-datum basis. Every decision has to be made in an atomic fashion, and the processed datum must be immediately available for later processing. *α-anonymity* does not require to buffer data and scales very efficiently. As such it is suitable for real-time deployments.

In *α-anonymity*, the first $\alpha - 1$ data subjects appearing in a ΔT- would have their quasi-identifier values removed, while the α-th subject would be the first one to have it visible. Yet - he/she belongs to a set of at least α data subjects - those $\alpha - 1$ be unknown. In a *k-anonymity* approach, all α quasi-identifiers would be made visible. In this sense, *α-anonymity* reduces the visibility of quasi-identifiers in the output stream.

VI. Related Work

Passive network monitoring threats users' privacy [21]. Because of that, we witness significant efforts to prevent information leakage from the network, and these efforts have been mostly centered around the deployment of encryption [22], [23]. For example, all newest web protocols by the time of writing (e.g., QUIC and HTTP/2) are built to run seamlessly over TLS. These initiatives reduce the amount of information exposed during the monitoring [24]. However, users' privacy can still be exposed in certain fields of Internet protocols. Server IP addresses and domains are two prominent examples, which may leak the sites visited by users. As such, those must be considered quasi-identifiers. Recent initiatives aim at encrypting plain-text domains seen in traffic, e.g., encrypting DNS [25] and Server Name Indications (SNIs) in TLS [26]. However, not all users will adopt these technologies any soon. In any case, those who monitor the network for legitimate reasons must also protect the privacy of all users, as mandated by regulations [4].

Several works propose techniques to anonymize traffic by obfuscating fields of protocol headers. The goal is to allow accurate network monitoring without threatening users' privacy. We can roughly group these techniques into (i) address anonymization and (ii) payload anonymization.

Address Anonymization: The simplest approach to achieve anonymization of IP addresses is the truncation of addresses. Everything, but the first n bits of the addresses (typically 8, 16 or 24), are set to zero. Truncation only partly mitigates the problem, as it is still possible to determine the subnet or the organization the truncated addresses belong to. More sophisticated techniques propose a prefix-preserving pseudo-anonymization, in which addresses are completely shuffled, but preserving the structure of subnets [27], [28], [29]. Crypto-PAN is perhaps one of the most popular prefix-preserving algorithms for IP addresses anonymization [14], [5]. The mappings between the original and anonymized addresses are determined by a passphrase and a symmetric block cipher. Here we rely on Crypto-PAN for IP address anonymization.

Payload Anonymization: Payload anonymization is more complex, as personal information may leak from different and complex protocols. Anonymization tools like `TCPdPriv` [30] and `TCPurify` [31] truncate TCP and UDP payloads, to remove all information contained in application layer protocols. This simple "reveal nothing" policy may lead to poor measurements. Other works propose sophisticated frameworks to handle specific application-level protocols. The authors of [32] remove sensitive information without affecting the payload. Packets are reconstructed into data stream flows, and application-level parsers modify the data streams as specified by a policy written in a high-level language. They provide limited anonymization primitives (constant substitution, sequential numbering, hashing, prefix-preserving, and adding random noise), forcing the user to write her own functions. The authors of [33] propose a programmable anonymization tool based on BPF filters, allowing the user to choose different actions according to the received protocol (IP, TCP, UDP, ICMP, HTTP or FTP).

Differently from these approaches, we explicitly target an operational deployment, in which anonymization must be achieved in real-time at tens of Gbps. Inspired by *k-anonymization*, we design a modular and flexible architecture to support *α-anonymity*. We focus on scalability and employ state-of-the-art packet capture techniques to make the system deployable on high-speed networks.

VII. Conclusion

We presented α-MON, a flexible and modular tool to anonymize network traffic according to a rich set of policies. We designed α-MON to be flexible and to provide anonymized traffic to multiple legacy monitors with different traffic visibility requirements. A key innovation in α-MON is the implementation of α-*anonymity*, a stream-based traffic anonymization technique that obfuscates quasi-identifiers that can be uniquely traced back to sets of data subjects. α-MON can search for them, for example, in domain names present in DNS, TLS and HTTP traffic.

We designed a scalable architecture and efficient data structures to implement α-*anonymity* at line-rate speed on multiple 10 Gbit/s links. α-MON reaches high throughput in typical scenarios with few CPU cores. Even in worst-case scenarios α-MON scales linearly with the number of cores, thanks to its design based on DPDK.

α-MON is available to the community as open-source software. Future work includes the extension of α-MON modules to cover other protocols, the development of mechanisms to find identifiers and quasi-identifiers in network traffic automatically as well as the analysis of the impact of α-*anonymity* on the operations of different legacy monitors.

Acknowledgments

The research leading to these results has been funded by both the Huawei R&D Center (France) and the EU Project PIMCity (Grant N. 871370). We would like to thank the Polito's IT staff for the feedback and support.

References

[1] L. Vassio, D. Giordano, M. Trevisan, M. Mellia, and A. P. C. da Silva, "Users' Fingerprinting Techniques from TCP Traffic," *ACM SIGCOMM Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 2017.

[2] G. Alotibi, N. Clarke, F. Li, and S. Furnell, "User Profiling from Network Traffic via Novel Application-Level Interactions," *International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016.

[3] A. S. Khatouni, M. Trevisan, L. Regano, and A. Viticchié, "Privacy issues of isps in the modern web," in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 588–594, 2017.

[4] European Commission, "REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (General Data Protection Regulation)," April 2016. Article 1, Subsection: 18, 23, 24, 28, 29, 30, 58.

[5] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme," *IEEE International Conference on Network Protocols (ICNP)*, 2002.

[6] P. Samarati and L. Sweeney, "Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression," *Technical Report SRI-CSL-98-04*, 1998.

[7] L. Sweeney, "k-anonymity: a Model for Protecting Privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

[9] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 106–115, IEEE, 2007.

[8] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 3–es, 2007.

[10] M. Trevisan, D. Giordano, I. Drago, M. Mellia, and M. Munafo, "Five years at the edge: watching internet from the isp network," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, pp. 1–12, 2018.

[11] Intel, "Data plane development kit." "https://www.dpdk.org/".

[12] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi, "Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 163–169, 2017.

[13] S. Woo and K. Park, "Scalable TCP session monitoring with Symmetric Receive-Side Scaling," *KAIST, Daejeon, Korea, Tech. Rep*, 2012. Accessed on 12/13/2016.

[14] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization," *ACM SIGCOMM Internet Measurement Workshop*, pp. 263–266, 2001.

[15] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, , and D. Rossi, "Experiences of internet traffic monitoring with tstat," *IEEE Network*, vol. 25, no. 3, pp. 8–14, 2011.

[16] "Benchmarking Methodology for Network Interconnect Devices." RFC 2544, Mar. 1999.

[17] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore, "Understanding pcie performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), p. 327–341, Association for Computing Machinery, 2018.

[18] T. Li and N. Li, "On the tradeoff between privacy and utility in data publishing," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 517–526, 2009.

[19] A. Meyerson and R. Williams, "On the complexity of optimal k-anonymity," in *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 223–228, 2004.

[20] J. Zhang, J. Yang, J. Zhang, and Y. Yuan, "Kids:K-anonymization data stream base on sliding window," *IEEE International Conference on Future Computer and Communication*, 2010.

[21] S. Farrell and H. Tschofenig, "Pervasive Monitoring Is an Attack." RFC 7258, May 2014.

[22] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the" s" in https," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 133–140, 2014.

[23] C.-l. Chan, R. Fontugne, K. Cho, and S. Goto, "Monitoring tls adoption using backbone and edge traffic," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 208–213, IEEE, 2018.

[24] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.

[25] P. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," Tech. Rep. 8484, RFC Editor, 2018.

[26] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood, "Encrypted Server Name Indication for TLS 1.3," Tech. Rep. draft-ietf-tls-esni-04, RFC Editor, 2019.

[27] R. Pang and V. Paxson, "A High-Level Programming Environment for Packet Trace Anonymization and Transformation," *ACM SIGCOMM*, pp. 339–351, 2003.

[28] M. Peuhkuri, "A Method to Compress and Anonymize Packet Traces," *ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001.

[29] M. Allman, E. Blanton, and W. Eddy, "A Scalable System for Sharing Internet Measurements," *In Proceedings of Passive —& Active Measurement (PAM*.

[30] G. Minshall, "Tcpdpriv," *http://fly.isti.cnr.it/software/tcpdpriv/*.

[31] E. Blanton, "Tcpurify," *https://isc.sans.edu/forums/diary/Truncating+Payloads+and+Anonymizing+PCAP+files/23990/*, 2008.

[32] R. Pang and V. Paxson, "A High-Level Programming Environment for Packet Trace Anonymization and Transformation," *ACM SIGCOMM Conference*, 2003.

[33] D. Koukis, S. Antonatos, D. Antoniades, E. P. Markatos, and P. Trimintzios, "A Generic Anonymization Framework for Network Traffic," *IEEE International Conference*, vol. 5, 2006.