α-MON: Traffic Anonymizer for Passive 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 at scale are essential for several applications, such as cyber-security, accounting and troubleshooting. They, however, threaten users' privacy by exposing sensitive information. Anonymization has been the answer to this challenge, i.e., replacing sensitive information with obfuscated copies. Anonymization of packet traces, however, comes with some challenges and 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 protocol fields according to flexible policies that cover all protocol layers. Beside classic anonymization mechanisms such as IP address obfuscation, α-MON supports z-anonymization, a novel solution to obfuscate rare values that can be uniquely traced back to limited sets of users. Differently from classic anonymization approaches, *z-anonymity* works on a streaming fashion, with zero delay, operating at high-speed links on a packet-by-packet basis. We quantify the impact of *z-anonymity* on traffic measurements, finding that it introduces minimal error when it comes to finding heavy-hitter services. We evaluate α-MON performance using packet traces collected from an ISP network and show that it achieves a sustainable rate of 40 Gbit/s on a Commercial Off-the Shelf server. 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 [1], [2]. 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 [3], [4]. Network measurements thus may expose privacy-sensitive information and shall be performed with care to avoid threatening users' privacy [5]. New privacy regulations (e.g., GDPR [6]) 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, since different network protocol fields represent different privacy threats. Client IP addresses are *identifiers*, i.e., they allow one to identify the users (devices) generating traffic immediately. As such, they must always be obfuscated. The classic approach is CryptoPAN [7], a method that replaces IP addresses with pseudo-encrypted copies while maintaining the network prefixes. Other protocol fields, while not carrying identifiers, still pose risks as they may help user reidentification, thus acting as *quasi-identifiers*. Server IP addresses and server names (e.g., in HTTP or TLS) can be quasi-identifiers. They give hints about users' interests and, in some cases, allow user reidentification. Quasi-identifiers, therefore, shall be obfuscated too.

Replacing all identifiers and quasi-identifiers in traffic measurements with obfuscated copies reduces the value of the traces substantially. 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, associating traffic to particular servers is instrumental, e.g., for network management, accounting and dimensioning.

Anonymization techniques like *k-anonymity* [8] can handle quasi-identifiers – i.e., obfuscating values that allow user re-identification. However, these approaches work with *batches* of data and are impractical with high-dimensional datasets, like, e.g., the set of websites users access. 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 – e.g., security monitors or passive meters.

α-MON follows a novel approach to anonymize packets on-the-fly. To this end, we introduce *z-anonymity*, a mechanism to hide infrequent field values (like unpopular server names) from the traffic. When observing a value in a data stream, *z-anonymization* removes it if less than z users share the value in the past ΔT time interval. Performing *z-anonymity* online requires ingenuity, and α-MON implements a scalable and parallel solution for this. We show that *z-anonymity* introduces minimal errors on volumetric traffic measurements, such as

the estimation of traffic share of popular web services and websites.

We evaluate α-MON performance on Common-Off-The-Shelf (COTS) hardware with traces collected from operational networks. We show that: (i) α-MON helps to protect sensitive data via *z-anonymity*, preventing the disclosure of field values associated with fewer than z users; (ii) α-MON allows most information that would be obfuscated by 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; (iv) α-MON introduces minimal errors on common measurement scenarios, e.g., allowing accurate accounting of the heavy-hitters' traffic. Finally, α-MON is publicly available as an open-source project.[1]

This paper extends our previous work [9] [2] on several directions. We implement additional functionalities to enable more flexible anonymization and enhance the experimental evaluation to study system parameters' impact: we have extended the implementation of *z-anonymity* to support generic protocol fields and many performance improvements, which will be discussed later. Moreover, we include a thorough analysis of the impact of *z-anonymity* on the typical traffic measurement accuracy. The reader can find in [10] an in-depth comparison between *k-anonymity* and *z-anonymity*. Using a probabilistic framework, we show that a proper choice of the z-anonymity parameters allows the data curator to obtain a k-anonymized dataset with reasonable certainty.

The paper is organised as follows. Section II defines *z-anonymity*, while Section III describes α-MON architecture, design and implementation. Section IV quantifies the impact of *z-anonymity* on traffic measurements. Section V benchmarks α-MON performance and shows how to tune system parameters. Section VI discusses the *z-anonymity* and α-MON approach and limitations. Section VII summarises the related work, and, finally, Section VIII concludes the paper.

II. z-ANONYMITY

The drastic increase in the rate at which personal data are collected has 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 [8], [11]. 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-anonymity has been extended with the *l-diversity* [12] and *t-closeness* [13] ideas, which we will discuss in Section VI.

k-anonymity however does not scale [14] and cannot be implemented with minimal delay. With α-MON we want to achieve *some* level of anonymity already during data collection, by hiding the most sensitive information observed in network measurements. This goal calls for highly scalable and zero-delay solutions. We lie in a scenario where anonymization must be performed in real-time and must scale up to multi-Gbit/s streams. Streaming approaches for anonymity [15], [16] load the incoming records in a structure and release anonymized data in batches, which is impractical with high-speed network traffic, given the large speeds of the input streams. In sum, we cannot assume to have the whole dataset, or a large subset of the data, at disposal for anonymization.

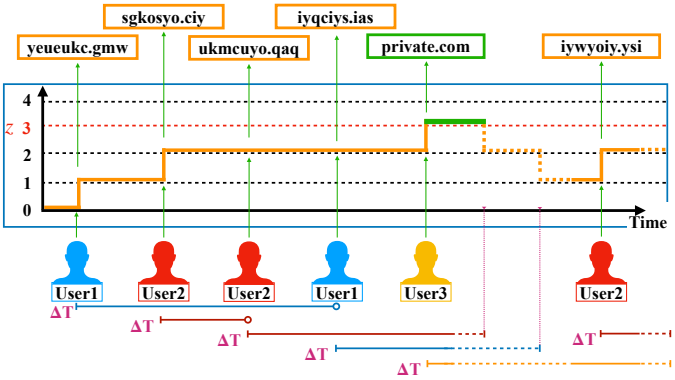


Fig. 1: *z-anonymity* concept. Three users access the FQDN `private.com` over time. When less than $z = 3$ unique users' are seen in the past ΔT, requests must be anonymized.

A. z-anonymity *definitions*

Here, we propose a novel concept of anonymity. We call it *z-anonymity*. It targets real-time, online processing, with minimum latency. In the following, we provide a formal definition. We assume that users are identified by an *identifier*. The most common identifier in network traces are client IP addresses[3].

Quasi-identifiers are attributes whose values must be controlled, as they may help to re-identify users. In our case, quasi-identifiers are fields present in protocol headers and payload that may be associated with a small group of users. Examples include specific server IP addresses and server names present in payloads (e.g., in DNS) and user-agent strings (e.g., in HTTP requests). For instance, an attacker could leverage a user's interest in a particular website to re-identify her and, in turn, all her traffic. *z-anonymity* aims at obfuscating rare values of quasi-identifiers in real-time, preventing these privacy attacks. We introduce the definition of *z-private quasi-identifier*.

Definition 1. A z-private quasi-identifier is a value observed at time t that is associated with less than z users in the past ΔT time interval.

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

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

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

[2] We decided to maintain the name α-MON since we consider α as a generic way to address *anonymization*. In *z-anonymity*, we decide to switch to z since it is only one of the possible anonymization technique implemented in α-MON, and that name could have been misleading.

[3] *z-anonymity* can handle any protocol field as an identifier.

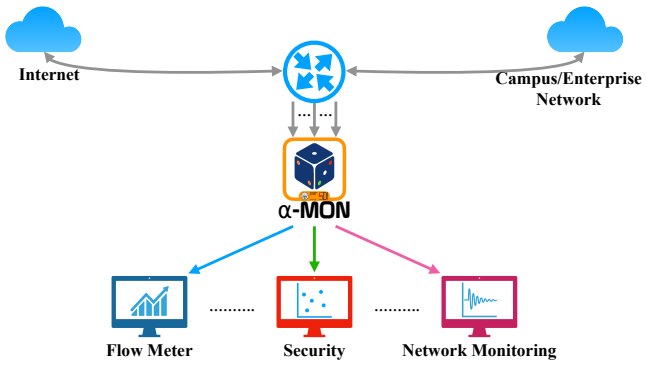


Fig. 2: Deployment scenario: α-MON anonymizes the traffic coming from a span port or an optical splitter and forwards it to different legacy monitors.

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

We exemplify the idea of *z-anonymity* in Figure 1. Here the quasi-identifiers are the Fully Qualified Domain Names (FQDNs) found in packet payloads, which refer to websites and web services. Suppose different users access the FQDN `private.com`. Let $z = 3$. The first four accesses shall be obfuscated as only two users accessed it up to then. When we observe User3's request, we have 3 users that have accessed `private.com` in the past ΔT. Thus, we allow User3's request to pass without anonymization. Notice that exposing `private.com` does not uncover User3, as attackers cannot 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, and their names would not be anonymized. Rare FQDNs that could bring specific information about users would likely be anonymized. In Section IV, we provide a thorough analysis of how *z-anonymity* impacts the accuracy of traffic measurements.

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 achieve *z-anonymity* at high-speeds.

A. Deployment scenario and requirements

Figure 2 shows the deployment scenario. α-MON operates as a classic network monitor, receiving packets from one or multiple network cards, either using 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. For example, a security monitor may need traffic with little modification, while a passive meter used for traffic accounting can operate with less information.

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

1) It must achieve *z-anonymity* to hide private quasi-identifiers with customizable z 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) [17], 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 in our previous work [18], 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 TCP and UDP flow status.[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 by instrumenting the network interface with a specific RSS hash seed [19].

Each thread receives a fraction of the overall traffic. According to custom-defined configurations, packets are replicated, their payloads anonymized, and, finally, 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. Traditional techniques for increasing system robustness to failures can be applied to α-MON. For example, it can be run in multiple machines for increasing reliability, as soon as traffic is steered accordingly (i.e., consistently sending packets of a flow to the same α-MON instance). In case of very critical setups, it would be possible to replicate two identical α-MON deployments by using multiple span ports or optical splitters.

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

C. Anonymization modules

We design α-MON to be modular and flexible. As such, the anonymization functions 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 receive original packets, while passive monitors 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 can apply *z-anonymity* on the found FQDNs, which are deleted from packets in case they are not z-private. α-MON can also apply *z-anonymity* on second-level domains – i.e., the FQDNs truncated after the top-level domain.[5] In this modality, α-MON releases the FQDN if not z-private. In case it is z-private, it checks if the second-level domain is not, and, in case, α-MON truncates the FQDN to the second-level domain. Similarly, any field of protocol headers could be subjected to *z-anonymity*, with customized z 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).

Layer 3: α-MON considers client IP addresses as identifiers and anonymize them using the CryptoPAN algorithm [20], [7]. 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 *z-anonymity*.

Layer 2: α-MON supports the removal of MAC addresses. Alternatively, as routers generally modify MAC addresses 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.

[5]For example `private.example.com` becomes `example.com`.

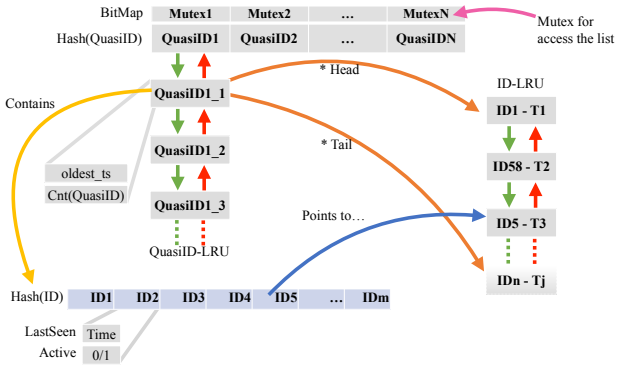


Fig. 3: Data structure used to handle quasi-identifiers.

D. z-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 *z-anonymity*. When processing a packet from a user identified by ID and containing the quasi-identifier $QuasiID$, α-MON must decide whether to keep $QuasiID$ or hide it. The decision is based on the counter *Cnt(QuasiID)* of users 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 3. As *z-anonymity* must globally count the number of users sharing each $QuasiID$, 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 users sharing the $QuasiID$, along with the timestamp of respective last occurrence. This *ID-LRU* is instrumental for purging those IDs 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 user 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 users sharing $QuasiID$ is increased. α-MON also goes through the $ID-LRU$ and deletes IDs older than ΔT. The *Cnt(QuasiID)* is decreased consequently.

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

Note that we need to purge from the data structure those entries older than ΔT. When accessing a *Hash(QuasiID)* bucket, α-MON expurges the expired entries in the *QuasiID-LRU* with a controllable probability P. This scheme limits extensive cleaning operations at each access. When cleaning *QuasiID-LRU*, α-MON controls the $ID-LRU$ for IDs older than ΔT. Again, the *Cnt(QuasiID)* is decreased consequently. If the counter indicates that the current $QuasiID$ is no longer in use, α-MON deletes it altogether.

Note that α-MON can 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 V-B.

E. Auxiliary data structures

α-MON implements an efficient structure to support per-flow management. This structure is instrumental for applying consistent anonymization decisions based on flow state – e.g., removing the payload of specific protocols (e.g., HTTP) or parsing 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 [21]. It builds on a hash-based data structure that provides $O(1)$ accesses to the per-flow metadata.

Given the current packet, the dedicated module performs a search in the flow table to verify the action performed previously: if the first packet of the flow has been subject to anonymization, the current one follows the same procedure. In the case of a new stream, α-MON creates the appropriate flow entry and checks how to anonymize the packet. The module includes a lazy garbage collection system for expired flows, similar to the one used in the *z-anonymity* module.

Finally, α-MON implements caching to speed up the anonymization of IDs, e.g., maintaining a per-thread cache of the anonymized IP addresses computed by CryptoPAN. In Section V, we show that this design allows α-MON to scale to several Gbit/s of traffic.

IV. The impact of *z-anonymity* on traffic measurements

We now quantify to what extent the traffic anonymized with the *z-anonymity* is useful to provide accurate network measurements. We aim to understand how common traffic analyses are affected when run behind an α-MON deployment. To this end, we perform a case study in which users (identified by client IP addresses) contact several hosts characterised by FQDNs (and second-level domains), which are considered quasi-identifiers.

We use a traffic trace collected on an operational network including more than 8 000 users who generate several millions of packets per second of traffic. The trace is three-days long, during which the users contacted 135 k (45 k) FQDNs (second-level domains). The FQDNs are present in TLS, DNS, and HTTP headers. We apply *z-anonymity* directly on FQDNs or on the corresponding second-level domains. Client IP addresses are used as identifiers. For these analyses, we implement an offline version of *z-anonymity* to process the traces, obtain statistics and show how these statistics vary with different parameters.

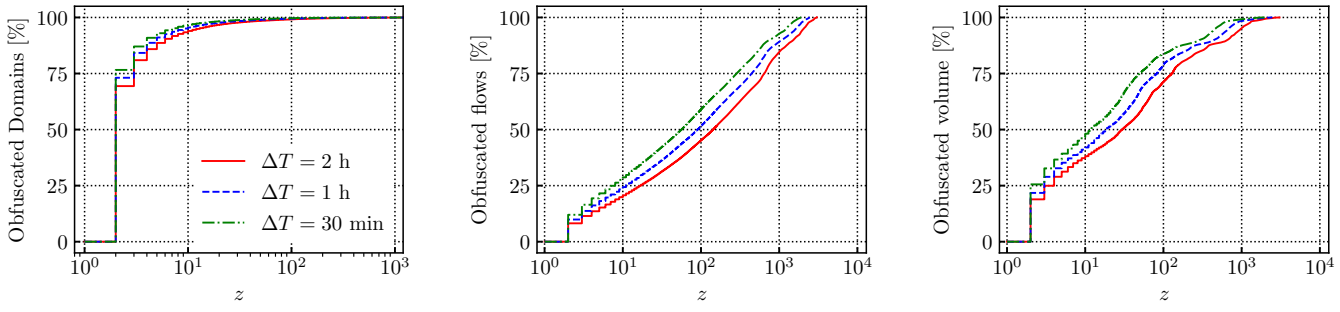
A. Anonymized volumes

We first analyze the fraction of traffic that *z-anonymity* would obfuscate when considering different values for z and ΔT. We show results in Figure 4 for the case of *z-anonymity* on FQDNs and in Figure 5 for second-level domains. First, consider the fraction of FQDNs that *z-anonymity* would obfuscate in Figure 4a. We notice that $z = 2$ already causes $\approx 75\%$ of the FQDNs to be obfuscated. When $z = 10$, the fraction increases to 90%. ΔT has a small overall impact. Similar considerations hold for the case of second-level domains (Figure 5a). Here, on the one hand, the coarser data granularity makes it more likely for a domain to pass *z-anonymity*. However, we find a smaller number of quasi-identifiers (45 k instead of 135 k), which balances the picture, allowing a similar share of domains to pass *z-anonymity*.

Different is the picture if we consider the number of flows (Figure 4b) and the byte-wise volume (Figure 4c) carried by flows for which the FQDN gets obfuscated. With $z = 2$, *z-anonymity* obfuscates the FQDN in only 10% of the flows, which account for $\approx 25\%$ of the traffic volume. Most of the obfuscated FQDNs are used by CDNs and include digits or random strings in the sub-domains. Taking instead the second-level domains as quasi-identifiers reduces the percentage of obfuscated bytes to negligible numbers for $z = 2$. This is caused by the nature of Internet traffic, where the majority of flows are directed towards a limited set of services [22].

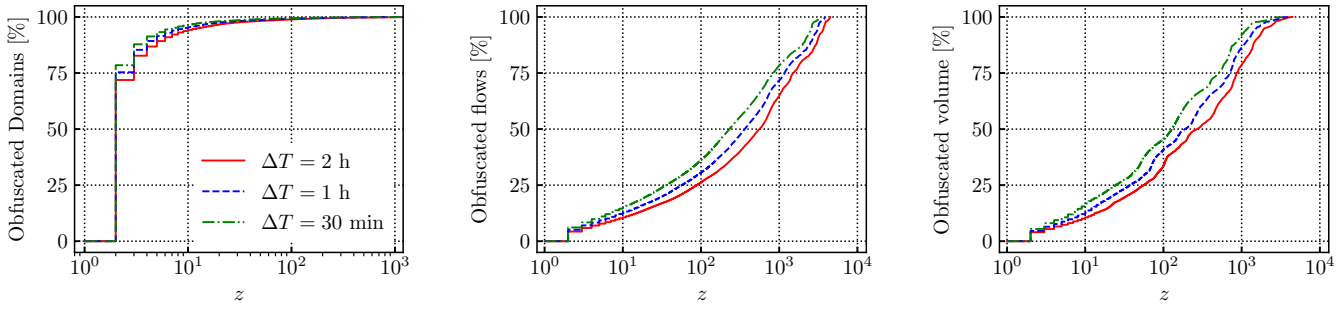
The impact of a large ΔT is more pronounced for high values of z, allowing a larger number of flows to avoid obfuscation. For example, if we set $z = 100$, a $\Delta T = 30min$ results in 60% of obfuscated flows; this fraction decreases to 52 $(46)\%$ if we set $\Delta T = 1h$ $(2h)$. If we consider second-level domains (Figure 5b and Figure 5c), we observe a similar picture. Considering the byte-wise volume, notice how the fraction of obfuscated traffic decreases, mainly due to the aggregation of CDN nodes and randomly generated domains to a single quasi-identifier.

In a nutshell, popular domains that carry little sensitive information are responsible for the majority of traffic. Letting their names in clear poses little threats for privacy, while



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

Fig. 4: Fraction of traffic obfuscated by z-*anonymity* with different values of z and ΔT. z-anonymized field: FQDN



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

Fig. 5: Fraction of traffic obfuscated by z-*anonymity* with different values of z and ΔT. z-anonymized field: Second-level domain.

still being very important for increasing visibility of network monitors. z-*anonymity* obfuscates the vast majority of FQDNs or second-level names that carry little traffic while allowing the popular names to be monitored.

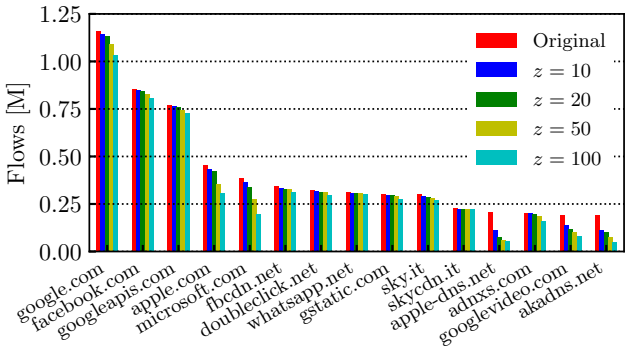
B. Impact of z-anonymity *on traffic accounting*

As a use case on accounting, we study how z-*anonymity* changes the traffic volume measured for the most popular services on the network. We assume that services can be identified by their second-level domains. In Figure 6a we show the number of flows for the top-15 services as measured on the original trace and after z-*anonymity* with different values of z. In this experiment, ΔT is fixed to 1 hour, and z-*anonymity* is applied to the FQDN.

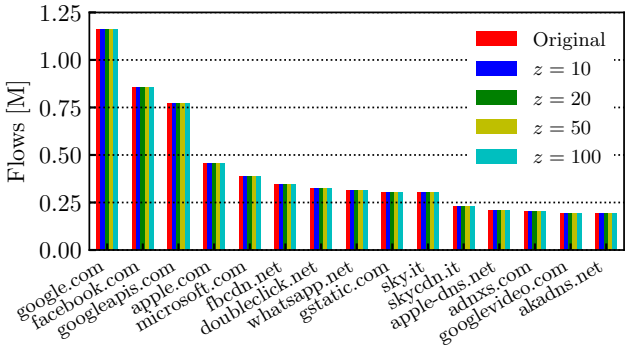
It is no surprise that the most popular service is *google.com*, followed by common services/platforms such as Facebook and Apple. The red bar represents the values measured on the original trace that we use as a baseline. If the traffic undergoes z-*anonymity*, the measured volume slightly decreases, but in almost all cases, the drop is limited to $10 - 15\%$ even with high values of z (yellow and cyan bar). In other words, z-*anonymity* would introduce a small measurement error in terms of traffic volume for these services, e.g., because less z users have requested their FQDNs in some ΔT (1 hour).

For a few cases, the difference is more pronounced; see, for example, *apple-dns.com*. In these cases, a single service/second-level domain holds a very large number of FQDNs, making them more likely to be anonymized. Indeed, in the case of *apple-dns.com*, we observe 1 044 subdomains, most of them differing uniquely for a two-digit code. Not shown in the figure, we observe a similar phenomenon for the CloudFront CDN, for which we observe 3 115 subdomains, each referring to a hosted website. In these cases, the measurement error introduced by z-anonymized increases. We can solve the issue running z-*anonymity* on the second-level domains directly – i.e., configuring α-MON to release the second-level domain if it passes the z-*anonymity* checks. With this setup the measured volume is practically equal to the original traces, as we show in Figure 6b. Here, the users' privacy is still preserved, as α-MON hides the sub-domains, releasing only the z-private second-level domains. Yet, the value of the anonymized traces is increased substantially.

We complete the above analysis with Figure 7, in which we broaden the bounds of pictures presented before. In the figure, we show with the red solid line the traffic volume (in terms of flows) for *all* services in the trace exceeding a minimum threshold of 1 000 flows (more than 1 000 names). They are sorted by popularity. The blue dashed line represents the volume as measured after z-*anonymity* with $z = 10$ and $\Delta T = 1$ h. We are interested in the deviation between the two lines, representing the measurement error. Considering z-*anonymity* on FQDNs (Figure 7), the error is minimal for the

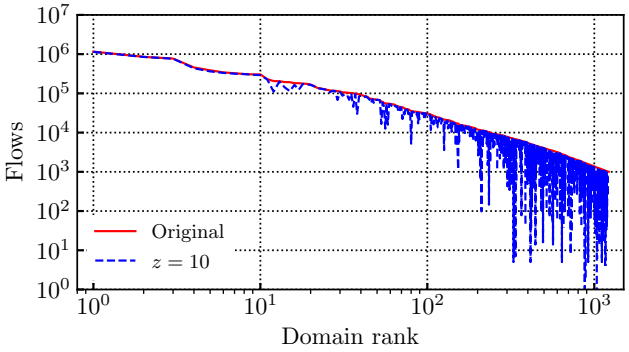


(a) *z-anonymity* on FQDN

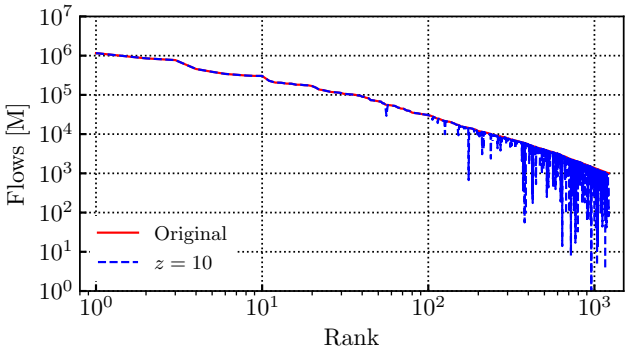


(b) *z-anonymity* on second-level domain

Fig. 6: Per-service volume measured from an z-anonymized trace.



(a) *z-anonymity* on FQDN



(b) *z-anonymity* on second-level domain

Fig. 7: Per-service flows measured on the original and on a $(z = 10)$-anonymized trace.

top-ranked domains, as already shown by the blue bars in Figure 6a. The deviation is still limited for the services with sizeable traffic, never exceeding an order of magnitude for those with at least (around) 10 k flows – top-200 names, left part of the figure. Clearly, the less frequent services are, the higher the chances α-MON anonymizes their FQDNs, thus increasing the measurement error. If we apply *z-anonymity* on second-level domains directly, Figure 6b shows that we obtain more reliable measurements also for less popular services. Also in this case, the error becomes high for infrequent services or those accessed by a very small number of users (see right side of the figure).

In summary, volumetric statistics are still reliable when targeting heavy-hitter services. For less frequent services, α-MON introduces a larger measurement error, as enforcing *z-anonymity* may lead to most occurrences of the associated FQDNs (or second-level names) to be obfuscated. By tuning z and ΔT, one can regulate the trade-off between privacy and data utility.

C. Load on α-MON data structures

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

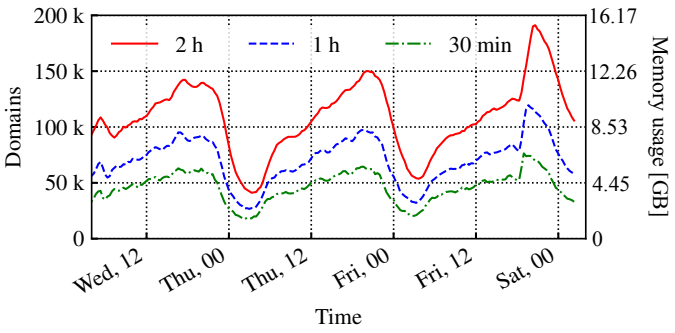


Fig. 8: Domains known by *z-anonymity* with different ΔT, in relation with the memory needed by α-MON.

Taking again FQDNs as an example, we consider the size of the set that *z-anonymity* must track – i.e., those FQDNs observed in the last ΔT interval. Figure 8 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 FQDNs are

	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.

seen in a two-hour interval – solid red line, see leftmost y-axis. No more than 100 000 (60 000) FQDNs appear with a ΔT of 1 hour (30 minutes). During the night, when traffic reduces, the number of active FQDNs 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. The rightmost y-axis of Figure 8 reports the memory footprint of the $QuasiID$ metadata, considering their actual size in our implementation. The memory usage is always below 17 GB for this setup.

Recall that the 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 domain names scales sub-linearly with the number of users. For example, during the peak hour, 1 000 (or 3 000) randomly selected users already contact 35 000 (or 60 000) FQDNs, while all 8 000 users contact 150 000 domains.

V. Performance Evaluation

We now evaluate the performance of α-MON in processing high speed traffic. We aim at evaluating how α-MON performance scales with the number of cores and the impact of different conditions, workloads and system parameters. We follow the standard benchmarking procedures defined in [23] for throughput tests.

A. Testbed and dataset

We instrument a 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. 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.[6] DPDK-Replay can loop over traces in memory, eventually replacing IP addresses on each pass, so to allow arbitrary benchmark duration.

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

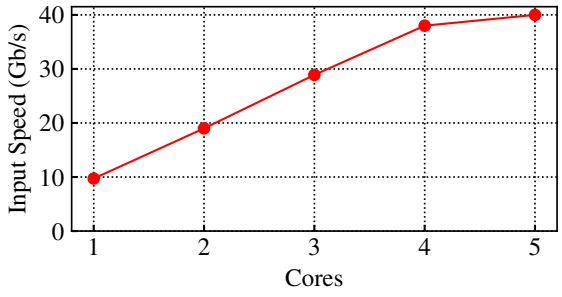


Fig. 9: Performance with four input and four output interfaces using the ISP-FULL trace.

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.

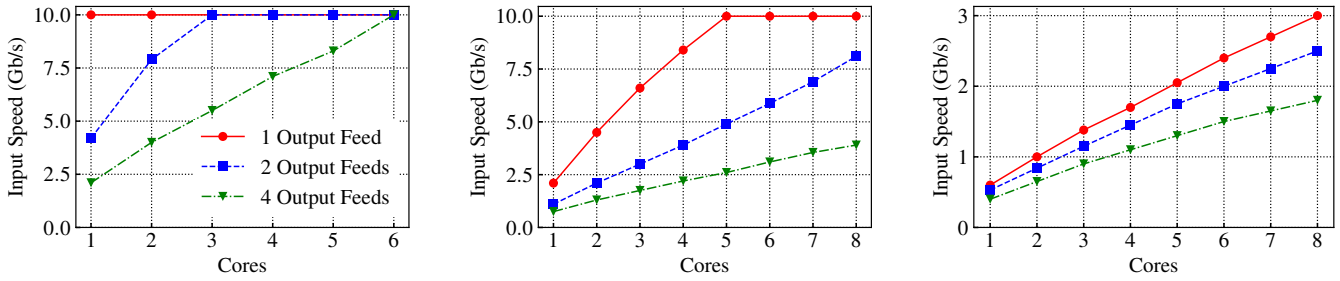
We process this trace to keep only up to TCP/UDP headers, removing payloads. This step results in a second packet trace – called ISP-HDR– in which packets are truncated to 54 B on average. We use this trace to benchmark the per-packet capture, processing and transmission speed of α-MON in a pessimist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

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

For each experiment, we seek the *throughput* [23], 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 $z = 10$ 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 8).

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



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

Fig. 10: Performance with different traces and consumer numbers.

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 to 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 9, 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 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 before hitting the PCI bus bandwidth limit [24].

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 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.

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 *z-anonymity* module for FQDN. 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 10. We report throughput for different traffic traces in separate figures, where lines indicate the number of output feeds. X-axes show the number of cores.

Figure 10a depicts the performance with the ISP-FULL trace. As already shown previously, 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: 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. Here, contention on the *Hash(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 10b - 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 *z-anonymity* module. In Figure 10c we see that throughput further decreases. Remind that packets undergoing *z-anonymity* generate updates on various data structures to track the set of users associated with each quasi-identifier. Moreover, parsing the payload to recover quasi-identifiers is time consuming too (e.g., to extract FQDNs in DNS payloads). Figure 10c shows that a single core sustains 0.6 Gbit/s with one output feed.

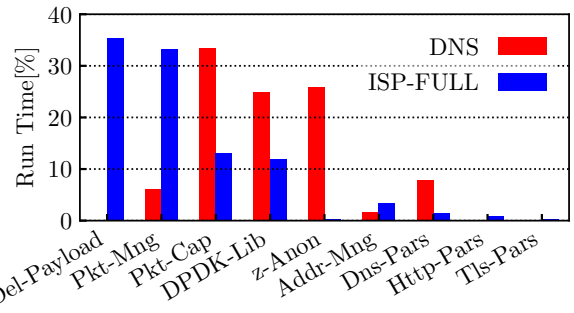


Fig. 11: Percentage of time spent on the most impacting modules.

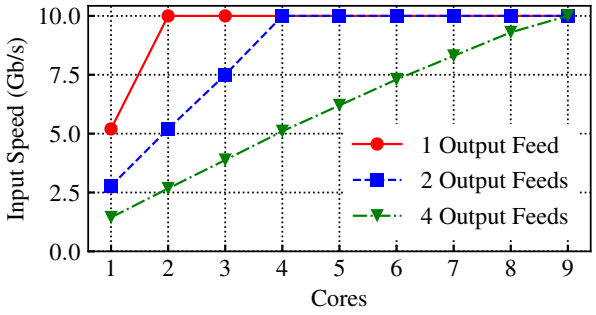


Fig. 12: Performance when applying *z-anonymity* on IP addresses and FQDNs (ISP-FULL trace).

Again, the throughput increases almost linearly with the number of cores, and eight cores can handle 3 Gbit/s of 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 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. α-MON scales linearly with the number of cores in all scenarios.

D. α-MON sub-module performance

We next show the impact on performance of α-MON components, by dissecting their execution time under different workloads. To this end, we instrument each α-MON sub-module with counters that use the CPU Time Stamp Counter to measure the elapsed time with a negligible performance penalty.[7] We then make experiments with the ISP-FULL and DNS traces, configuring α-MON with a single output feed, a single core, and replaying traffic at the sustainable rate – i.e., 10 Gbit/s for ISP-FULL and 0.6 Gbit/s for DNS.

Figure 11 shows the percentage of time spent on the main α-MON modules. We first notice how the nature of the traces determines different execution patterns. With ISP-FULL (blue bars), removing the payload from packets of insecure protocols (e.g., HTTP) absorbs most of the time due to a large number of memory write operations. Differently, with DNS (red bars), the *z-anonymity* module is invoked at each packet and accounts for 28% of the total execution time (it is less than 1% for ISP-FULL). The packet capture (with DPDK) and general management routines in both cases have a significant impact, larger for DNS due to the smaller size of packets (see Table I) and, thus, higher packet rate. Finally, note how IP address anonymization with CryptoPAN and protocol header parsing always have a marginal impact.

[7]The CPU Time Stamp Counter is a CPU register, thus, very fast to read.

E. z-anonymity *on other protocol fields*

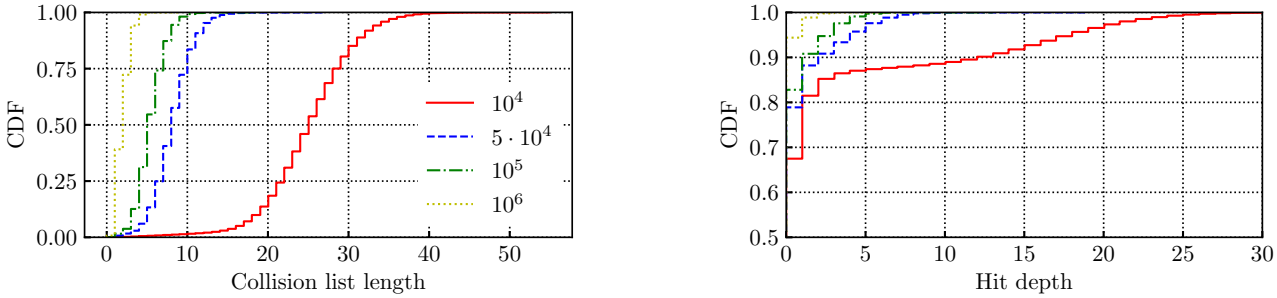
We now evaluate α-MON performance when applying *z-anonymity* on a wider range of protocol fields. Indeed, as described in Section III, the *z-anonymity* module is flexible and can operate on different protocol fields, from FQDNs contained in DNS, TLS and HTTP, to the IP addresses of the contacted servers. In this experiment, differently from the previous cases, we make use of this feature and configure α-MON to apply *z-anonymity* on both FQDNs and IP addresses.[8] This imposes a high load on data structures. Indeed, the *z-anonymity* hash table is loaded with additional $QuasiIDs$ and the flow hash table needs to be used to make consistent decisions on a per-flow basis.

α-MON performance slightly decreases, as we show in Figure 12, in which we report the sustainable rate with different numbers of output feeds. α-MON achieves line rate with 2, 4 and 9 cores with 1, 2 and 4 output feeds, respectively. When *z-anonymity* runs on FQDNs only (see Figure 10a), only 1, 3 and 6 are needed. In short, adding an additional field to *z-anonymity* decreases performance. As IP addresses are present in *every* single packet, decisions must be taken much more often than for FQDNs only. *z-anonymity* on IP address entails a $\approx$ 30% performance drop due to the higher number of operations.

F. *Tuning of the* z-anonymity *data structure*

Here we study the impact of the data structure size for the *z-anonymity* module. Indeed, our implementation builds on the (large) hash table *Hash(QuasiID)* used to accommodate the quasi-identifiers, and, for each quasi-identifier, the ordered list of associated users in the last ΔT. Collisions on the hash table are handled with lists, whose length should be kept as short as possible to avoid performance impairments. This section evaluates the impact of the hash table size on the list length and the time *z-anonymity* spends iterating on them. To this end, we run multiple experiments using the DNS trace and varying the hash table size. During the experiments, we record, for each access to the *z-anonymity* data structure, the length of the collision list (if any) and the position at which α-MON found

[8]α-MON applies *z-anonymity* on server IP addresses only, obfuscating internal client addresses with CryptoPAN.



(a) Length of the collision list for each QuasiID

(b) Depth of hits in the collision list

Fig. 13: Analysis of the behaviour of Hash(QuasiID) collision list. Lines represent different hash table sizes.

the matching quasi-identifier. The latter metric is particularly important given that α-MON handles collision lists in an LRU. As such, it is likely to find popular quasi-identifiers on the top of the list, avoiding exhaustive scans.

Figure 13a reports the distribution of the collision list length with different hash table sizes, from 10 k to 1 M elements. Clearly, a hash size much smaller than the number of quasi-identifiers leads to long collision lists. When the size is 10 k (red solid line), lists are 25-element long in median, but can reach up to 40 elements. Large hash sizes reduce the length of collision lists, and we notice that quasi-identifiers are uniformly distributed among all hash buckets (not shown in the figure). However, collisions happen by chance, and, even with a 10 M elements (yellow dashed line), we sporadically find a handful of collisions.

Fortunately, α-MON does not need to fully scan collision lists, as a searched quasi-identifier is usually much before the tail of the list. Only for still unknown items (a mismatch), the list must be scanned exhaustively to ensure the quasi-identifier is not already present. In Figure 13b, we report the distribution of the position in the list of the matching element for each access to the data structure. Comparing it with Figure 13a, we notice how in most cases α-MON does not scan the lists entirely. Even with a small 10 k hash size (red solid line), on 80% of cases the matching item is found on the first or second position, and in less than 5% the search goes further than the 20^{th} position. With large sizes, the probability of evaluating more than 10 list elements becomes negligible.

In summary, the hash table must be sized to the deployment scenario to prevent collision lists from growing excessively. If properly done, *Hash(QuasiID)* allows *O(1)* access, with reasonably short collision lists, thanks also to the LRU policy which saves exhaustive scans.

VI. Discussion on the z-anonymity approach

z-anonymity represents a new proposal for anonymizing sensitive information in network traffic. It shares with *k-anonymity*, *l-diversity* and *t-closeness* the idea that quasi-identifiers must be somehow controlled to prevent users' re-identification. No scheme can provide a guarantee of anonymity, and all schemes trade privacy with utility [25]. 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 [26] are not well-fit for real-time and online usage as they make decisions based on the *global distribution* of quasi-identifiers. Like traditional approaches, *z-anonymity* provides a trade-off and not full privacy guarantees. It however allows tuning the desired trade-off between privacy and data utility.

With *z-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. *k-anonymity* and similar approaches work on tabular data where the entire database (or a batch of data) 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 [27] – i.e., a system capable of applying *k-anonymity* on a stream database, where windows of data are considered. Other proposals [15], [16] work similarly, buffering data and releasing anonymized batches. Such approaches do not apply to our context since we cannot buffer lots of data while performing high-speed measurements in the network. Thus, we need to decide 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.

z-anonymity does not require to buffer data and scales very efficiently. As such, it is suitable for real-time deployments. To achieve that, *z-anonymity* is applied to each quasi-identifier in isolation as a performance trade-off. If the combination of multiple quasi-identifiers could lead to user re-identification, α-MON must be explicitly set to apply *z-anonymity* to the field combination. In fact, α-MON does not automatically search for such field combinations to increase performance.

Finally, in *z-anonymity*, the first $z-1$ user appearing in a ΔT would have their quasi-identifier values removed, while the z-th user would be the first one to have it visible. Nevertheless, she belongs to a set of at least z users, whose $z-1$ are unknown. In this sense, *z-anonymity* reduces the visibility of quasi-identifiers in the output stream.

VII. Related Work

Passive network monitoring threats users' privacy [31]. Because of that, we witness significant efforts to prevent infor-

TABLE II: Comparison of α-MON and alternatives.

	α-MON	ONTAS [28]	TCPdPriv [29]	TCPurify [30]
HW Implementation		✓		
SW Implementation	✓		✓	✓
Online Anon.	✓	✓	✓	✓
Stateful Anon.	✓			
L2	✓	✓		
L3	✓	✓	✓	✓
L4	✓		✓	
L5-7	✓			

mation leakage from the network, and these efforts have been mostly centered around the deployment of encryption [32], [33]. 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 [34]. However, users' privacy can still be exposed in certain fields of Internet protocols. Server IP addresses and FQDNs 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 FQDNs seen in traffic, e.g., encrypting DNS [35] and Server Name Indications (SNIs) in TLS [36]. However, not all users will adopt these technologies soon. In any case, those who monitor the network for legitimate reasons must also protect the users' privacy, as mandated by regulations [6].

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 [37], [38], [39]. Crypto-PAN is perhaps the most popular prefix-preserving algorithms for IP addresses anonymization [20], [7]. 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. Finally, in 2020 Kim *et al.* propose ONTAS [28], a flexible traffic anonymizer implemented directly in PISA-based programmable switches, which achieves high speed while anonymizing IP and MAC addresses.

Payload Anonymization: Payload anonymization is more complex, as personal information may leak from different and complex protocols. Anonymization tools like `TCPdPriv` [29] and `TCPurify` [30] 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 [40] 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 [41] 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 Gbit/s. Inspired by k-*anonymity*, we design a modular and flexible architecture to support z-*anonymity*. We focus on scalability and employ state-of-the-art packet capture techniques to make the system deployable on high-speed networks. Table II compares the features of α-MON against the three closest previous proposals described above, namely ONTAS [28], texttttTCPdPriv [29] and `TCPurify` [30], highlighting the novel capabilities of α-MON.

VIII. Conclusion

In this paper, 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 provide anonymized traffic to multiple legacy monitors with different traffic visibility requirements, from security monitors to simple passive meters. A key innovation in α-MON is the implementation of z-*anonymity*, a stream-based traffic anonymization technique that obfuscates protocol fields that can be uniquely traced back to a small sets of users. α-MON can search for them, for example, in the FQDNs present in DNS, TLS and HTTP traffic.

We designed a scalable architecture and efficient data structures to implement z-*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. We quantified the impact of z-*anonymity* on common traffic measurements, showing that it introduces negligible measurement errors. For example, if applied before accounting traffic of websites, only for very infrequent sites the measured values would substantially differ from correct values due to the anonymization.

α-MON is available to the community as open-source software. As privacy and privacy-preserving analytics are gaining momentum, we believe α-MON can help researchers, network administrators and practitioners maintain visibility on network traffic while preserving users' privacy at the same time. Future work includes the development of mechanisms to find identifiers and quasi-identifiers in network traffic automatically as well as the analysis of the impact of z-*anonymity* on the operations of different classes of legacy monitors.

ACKNOWLEDGMENTS

The research leading to these results has been funded by the Huawei R&D Center (France), the EU Project PIMCity (Grant N. 871370) and the SmartData@PoliTO center for Big Data technologies. We want also to thank the Polito's IT staff for the feedback and support.

REFERENCES

[1] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, 2019.

[2] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, D. Giordano, M. Mellia, and L. Venturini, "Selina: A self-learning insightful network analyzer," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 696–710, 2016.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] T. Favale, M. Trevisan, I. Drago, and M. Mellia, "α-MON: Anonymized Passive Traffic Monitoring," in *to appear in the 32th International Teletraffic Congress*, 2020.

[10] N. Jha, T. Favale, L. Vassio, M. Trevisan, and M. Mellia, "z -anonymity: Zero-Delay Anonymization for Data Streams," in *To appear in the 2020 IEEE International Conference on Big Data*, 2020.

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

[12] 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.

[13] 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.

[14] 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*, PODS '04, (New York, NY, USA), p. 223–228, Association for Computing Machinery, 2004.

[15] J. Li, B. C. Ooi, and W. Wang, "Anonymizing streaming data for privacy protection," in *2008 IEEE 24th International Conference on Data Engineering*, pp. 1367–1369, 2008.

[16] J. Cao, B. Carminati, E. Ferrari, and K. Tan, "Castle: Continuously anonymizing data streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, pp. 337–352, 2011.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia, "Five years at the edge: Watching internet from the isp network," vol. 28, pp. 561–574, IEEE, 2020.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] H. Kim and A. Gupta, "Ontas: Flexible and scalable online network traffic anonymization system," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, NetAI'19, (New York, NY, USA), p. 15–21, Association for Computing Machinery, 2019.

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

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

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

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

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

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

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

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

[41] 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.

BIOGRAPHIES

Thomas Favale comes from Brindisi and was born on April 1st, 1994. He got his Master Degree in Computer Engineering, specializing in the network branch, in 2019 at Politecnico di Torino. From May 2019 he started the Ph.D. at Politecnico di Torino under the supervision of Professor Marco Mellia, joining the Interdepartmental Centre for Smart Data. His research interests focus on traffic anonymization.

Martino Trevisan received his PhD in 2019 from Politecnico di Torino, Italy. He is currently an assistant professor (RTD-A) at the Department of Electronics and Telecommunications in the same university. He has been collaborating in both Industry and European projects and spent six months in Telecom ParisTech, France working on High-Speed Traffic Monitoring during his M.Sc. He visited twice Cisco labs in San Jose in summer 2016 and 2017, as well as AT&T labs during fall 2018. He was Visiting Professor at the Federal University of Minas Gerais in Brazil in 2019.

Idilio Drago is an Assistant Professor (RTD-b) at the University of Turin (UNITO), Italy, in the Computer Science Department. His research interests include cybersecurity, Internet measurements, artificial intelligence, machine learning and big data analytics. He particularly interested on how AI and data science approaches can help to extract knowledge from network monitoring traffic and help to automate network security. He was Visiting Professor at the Federal University of Minas Gerais in Brazil in 2019. He holds a Ph.D. in Computer Science from the University of Twente, the Netherlands, and a master's degree from the Federal University of Espirito Santo, Brazil. He was awarded an Applied Networking Research Prize in 2013 by the IETF/IRTF for my work on cloud storage.

Marco Mellia (F'21) is full Professor at the Department of Electronics and Telecommunications of Politecnico di Torino, Italy. His research interests are in the in the area of traffic monitoring and analysis, in cyber monitoring, and Big Data analytics. Marco Mellia has co-authored over 250 papers published in international journals and presented in leading international conferences. He won the IRTF ANR Prize at IETF-88, and best paper award at IEEE P2P'12, ACM CoNEXT'13, IEEE ICDCS'15. He is part of the editorial board of ACM/IEEE Transactions on Networking, IEEE Transactions on Network and Service Management, and ACM Computer Communication Review.