ONTAS: Flexible and Scalable Online Network Traffic Anonymization System

Hyojoon Kim
Princeton University
joonk@princeton.edu

Arpit Gupta
Columbia University
arpitgupta@cs.ucsb.edu

ABSTRACT

Access to packet traces is required not only to detect and diagnose various network issues related to performance and security, but also to train intelligent learning models enabling networks that can run themselves. However, packets in a network carry a lot of information which can be used to personally identify users and their online behavior. This requires network operators to anonymize packet traces before sharing them with other researchers and analysts. Existing tools anonymize packet traces in an offline manner, which incurs significant computational, storage, and memory overhead—limiting their ability to scale as the volume of the collected packet trace increases. In this paper, we present the design and implementation of an *O*nline *N*etwork *T*raffic *A*nonymization *Sy*stem, ONTAS, which can flexibly anonymize packet traces in the data plane itself using modern PISA-based programmable switches.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Network privacy and anonymity**.

KEYWORDS

Anonymization, Programmable switches, P4, PISA

ACM Reference Format:
Hyojoon Kim and Arpit Gupta. 2019. ONTAS: Flexible and Scalable Online Network Traffic Anonymization System. In *NetAI '19: ACM SIGCOMM 2019 Workshop on Network Meets AI & ML, August 23, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341216.3342208

1 INTRODUCTION

Conventionally, network operators have relied on manual diagnosis of network data for various tasks such as network behavior analysis, troubleshooting, forensics, anomaly detection, and so on. However, in recent years, the growing complexity of network management has pushed network operators and researchers to build *self-driving networks*, capable of autonomously detecting and resolving various network issues. Packet-level network streaming analytics systems (*e.g.*, Sonata [14], Everflow [33], dShark [31], Marple [19]) are the building blocks for such self-driving networks. These systems require continuous ingestion of network data in real time.

NetAI '19, August 23, 2019, Beijing, China

ACM ISBN 978-1-4503-6872-8/19/08…$15.00
https://doi.org/10.1145/3341216.3342208

Collecting network data poses a significant privacy threat. For example, vendor's tech support teams typically request packet traces while troubleshooting a network device. This data can be potentially used to identify network users and their online activities. Thus, network operators are required to collect network data in a privacy-preserving manner. In the past, practitioners and researchers have developed various tools for data collection (*e.g.*, `tcpdump`) and anonymization (*e.g.*, `tcpmkpub [20]`). Using these tools, network operators first collect and store raw packet traces and then apply the anonymization tool over the collected data in an offline manner. This approach not only results in wasteful usage of compute, memory, and storage resources but is also not capable of flexibly anonymizing network data for modern high-speed networks at line rate. This limitation impedes the development of self-driving networks that rely on collecting and analyzing network data at packet-level granularity in real time.

Network operators can perform online anonymization in different ways. One option is to apply the anonymization logic directly at the collector itself. However, this approach incurs a significant processing overhead and is hard to scale for high-speed networks. Another option is to offload the packet processing for anonymization to the network interface card (NIC) of the collector. Though this approach reduces the computational overhead, it will not work well with network streaming analytics systems that rely on programmable data planes to scale packet processing. In contrast, our system, ONTAS, leverages the flexible packet processing capabilities of modern *Protocol Independent Switch Architecture (PISA)* switches [5, 6, 15, 29] to anonymize packet streams in the data plane itself. ONTAS is scalable as it anonymizes packet streams at line rate for high-speed networks. ONTAS is also flexible as it supports a policy language for expressing anonymization tasks.

Building such a flexible and scalable anonymization system is challenging. First, it requires designing a policy interface that lets network operators express a wide range of anonymization policies without worrying about lower-level hardware details. Second, it requires developing a compiler that can translate the network operator's anonymization policies into the target-specific program. The compiler needs to make the best use of limited resources and ensure that switch's default forwarding behavior is unaffected by traffic anonymization.

We motivate the usage of online anonymization using PISA-based switches in Section 2. We present the design and implementation of ONTAS in Section 3. We use a testbed equipped with a state-of-the-art hardware switch and data collected from multiple enterprise networks to compare ONTAS' resource footprint with the state-of-the-art anonymization tools in Section 4. We present a more detailed comparison of ONTAS with the existing online

(and offline) anonymization tools in Section 5 before concluding the paper in Section 6.

2 BACKGROUND AND MOTIVATION

In this section, we first describe a use case where a researcher requires an anonymized packet trace. We describe the current workflow for providing this packet trace to the researcher, which is slow and inefficient. We then present our proposed workflow using ONTAS.

2.1 Motivating Use-Case

A network security researcher wants to identify different types of the Internet of Things (IoT) devices in a campus network and categorize them. She also plans to detect any unusual (malicious) network behavior, which is indicative of a possible compromise, of these devices [10]. To accomplish this, the researcher wants to first analyze all DNS requests coming from every end-host device in the target network to identify and classify different types of IoT devices. She hypothesizes that a specific IoT device class will have a uniquely identifiable DNS request pattern. Once the researcher has a list of clustered IoT devices, she plans to analyze traffic coming from each device in the list to detect any abnormal behavior.

To start this project, the researcher requests access to packet traces captured at various locations on the target network. The network operator can easily collect such packet traces. However, they can only share the data with her after anonymizing all personally identifiable information (PII), such as MAC or IP addresses. Network operators need to strike a balance between privacy and usability of the shared data. For example, a packet trace with all IP addresses zeroed out preserves privacy but is useless to the researcher. Each device's IP address should still be unique so that the researcher can distinguish one from another. The researcher also prefers to have the subnet or IP prefix information preserved. The MAC organizationally unique identifier (OUI) of each device is also valuable information that the researcher wants intact.

2.2 Current Workflow

To provide the requested data to the researcher, a network operator first needs to create spanning ports on routers or install tapping devices to generate mirrored traffic, which is then forwarded to a collector. The next step is to capture and store raw packet traces on disk. The campus network usually sees around a 3-4 Gbits/s traffic rate over the Internet in normal business hours. Therefore, capturing a packet trace for even ten minutes will result in 2400 Gigabits, or 300 Gigabytes of data. The network operator then applies a state-of-the-art anonymization tool over the raw packet trace to generate the anonymized version of the trace, which is also stored on disk. The network operator then transfers the anonymized packet trace to the customer's machine, where the researcher will finally be able to analyze the packet trace.

The current workflow for such a request relies on offline anonymization tools, such as `crypto-pan` [30], `pktanon` [13], or `tcpmkpub` [20]. These tools take raw packet traces as input and return their anonymized versions as output. Thus, these tools entail the overhead of redundantly storing both the original as well as the anonymized packet trace and redundantly processing (using both CPU and memory resources) all packets during data collection and anonymization. Some of these tools, such as `pktanon`, have the ability to anonymize packet traces in online mode. Yet, the tool's max anonymization throughput is around 57 Mbits/s even with simple randomization [13].

```
anonymize_multicast_broadcast: no
anonymize_srcmac_oui         : no
anonymize_srcmac_id          : yes
anonymize_dstmac_oui         : no
anonymize_dstmac_id          : yes
anonymize_srcipv4 : [172.17.1.0/24, 10.4.0.0/16]
anonymize_dstipv4 : [172.17.1.0/24, 10.4.0.0/16]
preserve_prefix              : yes
anonymize_mac_in_arphdr      : yes
anonymize_ipv4_in_arphdr     : yes
```

Listing 1: Anonymization policy example.

2.3 Proposed Workflow

Usage of offline anonymization tools incurs significant storage and compute overhead. Moreover, offline anonymization tools cannot provide continuous anonymized traffic feed to streaming analytics or inference systems in real time since such tools cannot keep up with the speed of live traffic feed. In contrast, we propose a new workflow that anonymizes a packet stream, at line rate, in the data plane itself.

Historically, the use of fixed protocols for transforming raw data packets in the network prevented network operators from performing anonymization in the data plane. However, recently proposed *Protocol Independent Switch Architecture (PISA)* switches provide many features that permit the implementation of customized packet processing logic in the data plane [5, 6, 15, 29].

Here, ONTAS translates a network operator's anonymization policy into a target-specific program (*e.g.*, P4 [3]). This program synthesizes match-action tables in the data plane which anonymize the incoming packet stream at line rate. Such capability opens up the possibility of building much-desired network streaming analytics systems that can process packets at line rate in a privacy-preserving manner.

3 ONTAS' DESIGN AND IMPLEMENTATION

In this section, we describe the design and implementation of ONTAS that uses PISA switches for enabling online anonymization.

3.1 Expressing anonymization policies

PISA switches can be programmed using the P4 language. Unlike higher-level programming languages supported by an x86 server (*e.g.*, Python), the P4 language operates at a relatively lower level of abstraction. Requiring a network operator to directly express their anonymization policies as a P4 program increases the participation threshold. Not only will it necessitate learning a new programming language, but will also require them to take various low-level resource constraints (*e.g.*, number of physical stages, number of actions in a stage, etc.) into consideration [14]. Also, it

will require them to compose their anonymization policies with forwarding—making the data plane more brittle and error-prone.

ONTAS abstracts away the lower-level hardware details from network operators for expressing their anonymization policy. To make it easy and straightforward, ONTAS allows an operator to specify a custom anonymization policy in a simple `(option:value)` formatted configuration file. To make ONTAS as flexible as possible, we implement essential anonymization features noted by previous works [20, 30] as well as the best practices list by CAIDA [8]. ONTAS currently does not support anonymizing layer 4 (TCP/UDP) fields. We believe extending ONTAS' policy interface to support these fields is incremental and leave this for future work (Section 6).

Listing 1 shows the configuration file for an example anonymization policy. Here, network operators specify which packet fields they want to anonymize. It shows the options in bold. First, the network operator specifies whether to anonymize multicast and broadcast packets. Special Ethernet addresses are reserved for multicast or broadcast traffic, thus are not unique for each device or user. This gives an option to operators to preserve such information if desired. We support four different options for anonymizing the Ethernet addresses. Each option anonymizes the first or the last 24 bits of the source or destination mac address. The first 24 bits identify an Organizationally Unique Identifier (OUI) and the last 24 bits identify a unique host (interface). By separating the processing of the MAC OUI and ID field, ONTAS enables anonymization of each field, independently. The next set of options hash the source and destination IP addresses in a prefix-preserving manner for the given set of prefixes. To not preserve the prefix, an operator can enter `no` for the `preserve_prefix` option. The last set of options specify anonymizing the sender and target MAC and IP addresses in ARP packets.

3.2 Compiling Anonymization Policies

3.2.1 Abstract Packet Processing Model. On PISA switches, a reconfigurable parser constructs a packet header vector (PHV) for each incoming packet. The PHV contains not only fixed-size standard packet headers but also custom metadata for additional information. A PISA switch has a fixed number of physical stages, each processing the PHVs in sequence. The packet processing pipeline is a sequence of custom match-action tables implemented using match-action units (MAU) in each stage. Each MAU applies stateless or stateful action over PHVs. Finally, a deparser serializes the transformed PHV into a packet before sending it to an output port.

Online anonymization requires executing a directed acyclic graph (DAG). Here, each node in the DAG performs the obfuscation operation corresponding to an option in the configuration file. This requirement aligns well with PISA's packet processing model. Anonymization-related operations for each option can be applied using a set of match-action tables. Tables for different options can be combined to execute an anonymization policy on packets. We now describe how ONTAS leverages this observation to anonymize network traffic directly in the data plane.

3.2.2 Compiling Options. Compiling anonymization policy to PISA switches requires translating the DAG of obfuscation options into an equivalent DAG of match-action tables. We will now focus on how to map individual obfuscation options into an equivalent set of match-action tables.

Multicast or broadcast. This option requires identifying whether a packet is a broadcast or multicast, and tag them for anonymization if set as `yes`. These operations can be executed using a single match-action table in the data plane and adding a metadata `skip`. The match-action table inspects the least-significant bit of the first octet in the packet's destination Ethernet address. If the bit is set, then the packet is identified as multicast or broadcast traffic and, depending on the anonymization policy, the `skip` field is set to zero or one. No further obfuscations are applied on packets with `skip` field set to one.

Ethernet addresses. As discussed earlier, we support four different options for anonymizing the Ethernet address. To enable these options, ONTAS parses the first and the last 24 bits of the two Ethernet addresses separately. The parser then stores the parsed values as metadata for further processing. Each of these options is implemented using a single match-action table. If the option is enabled, the match-action tables apply a hashing operation to update the value of the corresponding metadata field.

IP addresses. ONTAS supports prefix-preserving anonymization for source and destination IP addresses. This option first requires determining the prefix length (l) and then obfuscating the remaining ($32 - l$) bits of the IP address. To enable prefix-preserving anonymization, we divide packet processing into two logical stages. The first stage uses a match-action table to determine the prefix length. This table updates the prefix length metadata field of the packet and then stores the prefix-part and the remaining part of the IP address separately. This enables ONTAS to only hash bits in the remaining part while leaving the prefix part intact. For example, for the policy in Listing 1, this table will match on prefixes `172.17.1.0/24` and `10.4.0.0/16` and update the prefix length as 24 and 16 for the matching packets respectively. The second stage uses a match-action table to apply the hash function over the source (or destination) IP address' remainder ($32 - l$) bits and store the updated value in the packet's metadata. Thus, anonymizing IP addresses in prefix preserving manner requires a total of four match-action tables, two for each source and destination IP address fields. Finally, ONTAS recalculates and updates the IPv4 header checksum value using the post-anonymization IP addresses.

ARP packets. ARP request and reply packets contain sender and target MAC and IP addresses in the ARP header, which is separate from the MAC and IP address in the Ethernet and IP frame itself. In ONTAS, ARP packets are identified at the parser by inspecting the EtherType `0x0806`, which is a two-octet field in an Ethernet frame. ONTAS then parses ARP packets and locates the sender and target MAC and IP addresses in the ARP header that should be hashed. Here, instead of having a separate Ethernet and IP address anonymization policy for ARP packets, ONTAS follows the policy that the operator already specified for multicast, broadcast, Ethernet address, and IP address for the previous stages. For example, if the operator wrote a policy to preserve the multicast broadcast address, preserve the MAC OUI, and preserve IP prefix, the same policy will be applied to the sender and target Ethernet and IP addresses in the

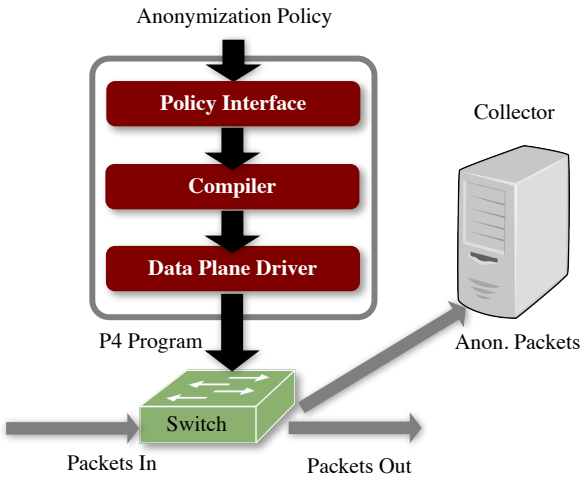

Figure 1: ONTAS architecture.

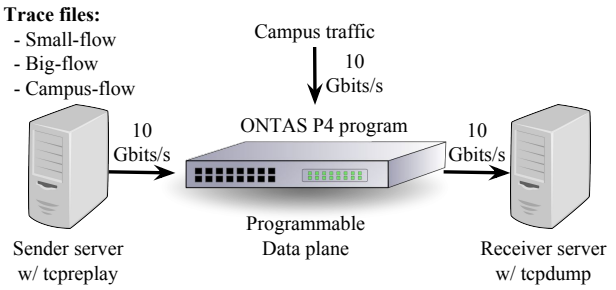

Figure 2: Testbed for ONTAS online anonymization.

Traces	Size	Packets	Throughput	Duration
`Small-flow` [25]	9.4 MB	14,261	247 Kbits/s	5 mins
`Big-flow` [25]	368 MB	791,615	9,477 Kbits/s	5 mins
`Campus-flow`	23 GB	25,864,345	3,169 Mbits/s	1 min

Table 1: Packet traces used for evaluation.

ARP header. Of course, an operator can also opt out of hashing the sender and target addresses in ARP packets altogether by putting `no` for the respective option in Listing 1. ONTAS will then not install flow table entries for this stage.

3.2.3 Compiling Policy. We now focus on compiling the anonymization policy by composing individual obfuscation operations into a DAG of match-action tables such that it reports anonymized packet streams to the collector without affecting the original packet's forwarding behavior.

Composing packet-processing pipeline. In addition to mapping individual options to match-action tables, ONTAS needs to synthesize the resulting data-plane mappings in a way that respects the ordering between various options. For example, the match-action tables for the multicast and broadcast option should be applied before the IP and Ethernet options. For the set of options that do not require strict ordering between them, ONTAS synthesizes a pipeline of match-action tables in the data plane without enforcing any particular order.

Preserving packet's forwarding behavior. ONTAS preserves packet forwarding decisions by hashing only the policy-specific metadata fields, rather than the packet contents that might affect forwarding decisions (*e.g.*, IP address, Ethernet address, etc.). For each option, it only updates the value in the metadata fields—ensuring that the original packet field values are not affected by anonymization operations. After applying all the anonymization match-action tables for various options in the policy, ONTAS applies a match-action table which clones the original packet and updates various packet fields with their anonymized version before sending it to a output port, *i.e.*, SPAN port, towards a collector. Updating the packet fields only for the cloned copy ensures that online anonymization does not affect the original packet's forwarding behavior.

3.3 Prototype Implementation

ONTAS has three main components: (1) policy interface, (2) compiler, and (2) data-plane driver (see Figure 1). The policy interface lets network operators specify their anonymization policy as a configuration file. Listing 1 shows an example of how network operators can express anonymization policies using ONTAS.

The compiler, implemented in Python, takes this configuration file as input to generate (1) a P4 program for configuring match-action tables in the data plane, and (2) a set of commands to configure match-action rules in the data plane. To generate the P4 program, the compiler first analyzes the configuration file to identify the required set of packet and metadata fields. It uses this information to configure the packet parser for the PISA switch. It then generates the necessary P4 program to translate individual options into a set of match-action tables. Finally, it takes the partial ordering between different options to synthesize the final packet processing pipeline. As an example, realizing an anonymization policy expressed using ten options generated 833 lines of P4 code.

The data-plane driver, implemented in Python, facilitates communication between ONTAS and data-plane targets. The ONTAS currently has drivers for two PISA switches: the BMV2 P4 software switch [28], which is the standard behavioral model for evaluating P4 code; and the Barefoot Wedge 100BF-32X (Tofino) [11], which is a 6.4 Tbits/s hardware switch. The data-plane driver currently uses Apache Thrift RPC [4] to remotely interact with the PISA switch.

4 EVALUATION

In this section, we demonstrate the scalability of ONTAS compared to state-of-the-art anonymization tools. More concretely, we show that ONTAS eliminates the post-processing time spent on anonymizing packet traces, which can take up to 15 minutes for one-minute campus traffic, and reduces the required disk space by half. We also demonstrate that online anonymization of packet fields in the data plane does not come at the cost of correctness.

4.1 Setup

Packet traces. To demonstrate the performance of ONTAS under different workloads, we use three different packet traces for evaluation: `Small-flow` [25], `Big-flow` [25], and `Campus-flow`. Table 1 shows various attributes, such as the number of packets, duration, etc., for each trace. The `Small-flow` trace is synthetically generated by combining packet traces of multiple applications utilizing different protocols at a relatively low data rate. The `Big-flow` trace is collected from a busy private network's access point to the Internet. Compared to `Small-flow`, `Big-flow` has a higher data rate and a smaller average packet size. The `Campus-flow` trace is collected between the Internet and Princeton University.

Comparison to existing tools. We compare ONTAS' performance with three state-of-the-art anonymization tools: `tcpmkpub` [20], `scrub-tcpdump` [32], and `pktanon` [13]. To quantify performance difference, we only selected tools that offer similar (or better) flexibility as ONTAS. We provide a further description of each tool and the flexibility they offer for packet anonymization in Section 5.

Testbed. Figure 2 shows our testbed setup. We use Edge-core Wedge 100BF-32X with Barefoot's Tofino chip [11] as the data-plane target . This switch can handle up to 100 Gbits/s traffic over 32 physical ports. We use `tcpreplay` [25] at the sender server to replay packet traces towards the data plane. We also fed a real-time packet stream from campus network to the Tofino switch to simulate a real use-case scenario. The switch mirrors the original packets and sends the anonymized version of them to the span port connected to the collector running `tcpdump`. To avoid packet drops, we use NIC tuning techniques [12] and `PFRING` [22] at the collector. We use a well-provisioned server (2 x Intel Xeon Silver 4114 with 2.2GHz CPU, 40 cores, 96 GB memory, and 240 GB storage) to quantify the performance of offline anonymization tools.

4.2 Quantifying Overheads

We will now compare the packet processing and storage overheads for ONTAS with the other anonymization tools.

Packet-processing overhead. ONTAS anonymizes packet stream at line rate ($O(ns)$) in the data plane, *i.e.*, it entails 0% packet processing overhead. In contrast, the state-of-the-art tools operate over collected packet traces in an offline manner. Figure 3 shows the time taken by these tools for completing anonymization of Ethernet address, IP address, and ARP header field against the three packet traces. We also quantify the packet processing overhead for `tcpmkpub` in the *speculative* mode which trades accuracy for faster packet processing. We report the elapsed time as average over ten runs for each trace-tool pair.

As expected, the packet processing time increases with the number of packets [1]. The `tcpmkpub` tool running in speculative mode (`tcpmkpub_s`) shows slightly better performance. Among the offline tools, `scrub-tcpdump` is the fastest as it skips the Ethernet layer and only anonymizes the IP and TCP headers. For `Small-flow` and `Big-flow` traces, all tools take less 20 seconds (7% overhead) to complete. For the `Campus-flow` trace, however, `scrub-tcpdump` takes around two minutes (200% overhead) while others take around 15 minutes (1500% overhead) to complete the anonymization.

[1] `tcpmkpub` tool failed to anonymize all packets of the `Big-flow` trace. We are currently investigating the root cause of this error.

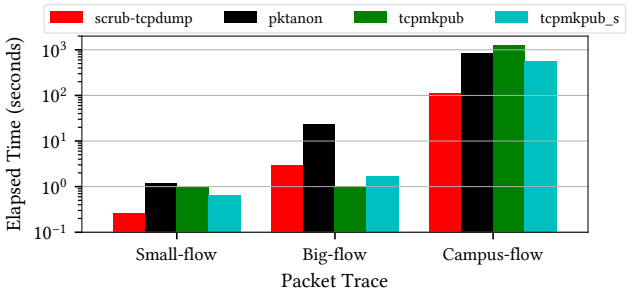

Figure 3: Elapsed time (in log-scale) when anonymizing packet traces using existing tools. `tcpmkpub_s` denotes `tcpmkpub` tool running in the speculative mode.

Storage overhead. The storage overhead for offline anonymization tools is double than that of online ones. They require additional storage to store raw packets before anonymization. As we discussed earlier, existing online anonymizations do not scale for high-speed networks. Thus, network operators are currently forced to incur this additional storage overhead. For example, an one-minute packet trace, collected from campus network operating at a modest data rate (3 Gbps), requires 23 GB storage space. Training learning algorithms for the self-driving networks will need capturing packet traces for multiple hours, which translates to TBs of storage overhead. Thus, reducing the disk space requirement by half, by using ONTAS, is a significant saving as the rate and size of a packet trace increases.

4.3 Validating Correctness

Line-rate anonymization of a packet stream in the data plane does not sacrifice correctness. To validate that ONTAS correctly applies different anonymization policies, we compared the packet traces before and after anonymization for different policies. For brevity, we only show the results using the `Small-flow` trace.

To compare the two packet traces, we compare packets from two files in chronological order. For each packet pair, we detect if they differ and if so, for which field (*e.g.*, IP address, MAC address, etc.). Note that the `Small-flow` trace has 14,261 packets in total. For a given policy, we report the total number of pairwise distinct values for each packet field. We present the results of this analysis for four different anonymization policies.

Policy-1. This policy should only anonymize source MAC addresses but preserve the OUI information. Table 2 shows that only the source MAC address differs between the two traces. Our script also confirms that ONTAS preserves the OUIs for this policy.

Policy-2. This policy should anonymize both source and destination MAC addresses, but not broadcast and multicast packets. Table 2 shows that out of total 14, 261 packets, 14, 202 have distinct MAC address between the two traces. The difference of 59 packets is attributable to multicast and broadcast packets.

Policy-3. This policy augments policy-2 by additionally anonymizing all source and destination IP addresses. It does not anonymize these fields in a prefix-preserving manner. Table 2 shows that out

	Src Mac	Dst Mac	Src IP	Dst IP
Policy-1	14,261	0	0	0
Policy-2	14,261	14,202	0	0
Policy-3	14,261	14,202	14,243	14,233
Policy-4	14,261	14,202	686	860

Table 2: Validating correctness. For a given policy, it shows the total number of pairwise distinct values for each packet field.

of total 14, 243 IPv4 packets, we observe 14, 243 distinct source and 14, 233 distinct destination IP addresses between the two traces. The difference of 10 packets is attributable to IPv4 broadcast packets.

Policy-4. The goal of this policy is similar to policy-3, except that it anonymizes IP addresses belonging to prefix `10.0.0.0/16` in a prefix-preserving manner. The original trace has 686 and 860 distinct source and destination IP addresses for prefix `10.0.0.0/16`. Thus, the result in Table 2 validates that ONTAS anonymized the data in a prefix-preserving manner correctly.

5 RELATED WORK

Offline anonymization. Crypto-pan [30] is a well-known tool for anonymizing IP addresses in a prefix-preserving manner and is widely-used by various institutions including CAIDA [9]. TCPAnon [24] can obfuscate fields in the application layer such as HTTP, SMTP, POP3, IMAP4, and FTP. However, the tool can only replace them with a constant value. Tcpmkpub [20] supports anonymization of IP (prefix-preserve), Ethernet addresses (OUI-preserve), and various fields in ARP, ICMP, UDP, and TCP. ONTAS supports most of the Tcpmkpub features. However, unlike these offline tools that only operate over stored raw packet trace files, ONTAS anonymizes packets in the data plane itself at line rate.

Online anonymization. Tcpurify [26] is a packet capture tool similar to tcpdump, but only anonymizes the IP address with random permutation and truncates the payload before writing to disk. Anonyflow [18] is an in-network IP address anonymization system that uses OpenFlow switches [17]. Anonyflow's goal is a bit different; it hides users' identity while they use the network, similar to what onion routing [23] (*e.g.*, Tor) achieves. Scrub-tcpdump [32] can do random permutation on IP addresses, TCP/UDP ports, TCP sequence number, TCP flags, TTL, packet length, and so on. It can run in both offline and online mode. The tool, however, cannot anonymize MAC addresses. PktAnon [13] is an offline and online anonymization tool that can obfuscate ARP, Ethernet, ICMP, IPv4, IPv6, TCP, and UDP fields. It is also possible to select different hashing algorithms for each field. The tool is rich in anonymization features and can run in online mode. However, unlike ONTAS, the processing performance is around 310 Mbit/s for just overwriting fields with a constant value, and simple randomization degrades it further down to 57 Mbit/s.

6 CONCLUSION AND FUTURE WORK

In this paper, we present ONTAS, an online anonymization solution that can obfuscate major personally identifiable information (PII), such as Ethernet and IP addresses, from a packet stream. We leverage the recent development of programmable data planes, such as Barefoot Tofino [27], for anonymizing a packet stream at line rate. Streaming analytics systems are the building blocks for self-driving networks [2], but their inability to process network traffic in a privacy-preserving manner has been a roadblock for their adoption. ONTAS's ability to anonymize packet streams at line rate opens up the vast opportunities for building streaming analytics systems without compromising privacy, which in turn will catalyze research in the area of self-driving networks.

ONTAS prototype lacks some features, such as anonymizing TCP/UDP field values. This is because our first and foremost focus was on PII fields, such as IP and MAC addresses. That said, as TCP/UDP fields can be used to identify certain device types or applications used in the network [20], we plan to extend ONTAS to support anonymizing them. As ONTAS' resource usage footprint is minimal, we believe such an extension will be trivial.

Currently, ONTAS does not support concurrently applying multiple privacy policies. It only applies one privacy policy at a time. For example, it is not possible to express, "apply policy 1 for traffic with the source IP address in the 192.168.1.2/24 subnet, and policy 2 for the rest." Given the flexibility of PISA switches, we believe extending ONTAS to concurrently apply multiple privacy policies and express predicates for each policy is trivial. We leave this extension as one of our future works.

Currently, ONTAS prototype uses a built-in, outdated hashing algorithm: Cyclic Redundancy Check 32-bit (CRC-32) [16, 21]. CRC-32 is meant to calculate checksums and is not a secure cryptographic hashing algorithm. The P4 language supports adding a salt to the algorithm, making it harder to reverse the hashed values. However, support for a better hashing algorithm in PISA-based switch is much more desirable. We believe this limitation is not fundamental. Recent work demonstrates implementing more secure hashing algorithms using existing resources [1]. The latest version of the P4 language [7] supports usage of external cryptographic hashing algorithms. In the future, we plan to extend ONTAS to use more secure hashing algorithms for anonymization.

We believe integrating ONTAS with streaming analytics system such as Sonata [14] is a promising future direction. Such integration will enable flexible network monitoring without compromising the privacy of network users at scale. This integration will entail (1) providing a unified programming abstraction to simplify expressing and composing anonymization policies with monitoring queries, and (2) designing an efficient compilation algorithm to make the best use of limited network resources.

REFERENCES

[1] AES encryption P4 implementation. https://github.com/chenxiaoqino/p4-projects/tree/master/AES.p4app.
[2] Workshop on Self-Driving Networks-Report. https://nsf-srn-2018.cs.princeton.edu/nsf-srn-report.pdf.
[3] The P4 Language Specification Version 1.0.5. https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf, November 2018.
[4] Apache Thrift API. https://thrift.apache.org/.
[5] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. *ACM SIGCOMM*, 2013.
[6] Gordon Brebner. P4 for an FPGA Target. In *P4 Workshop*, 2015. https://schd.ws/hosted_files/p4workshop2015/33/GordonB-P4-Workshop-June-04-2015.pdf.

[7] Mihai Budiu and Chris Dodd. The p416 programming language. *Operating Systems Review*, 51(1):5–14, 2017.
[8] CAIDA: Summary of Anonymization Best Practice Techniques. https://www.caida.org/projects/predict/anonymization/.
[9] CAIDA: Data Collection, Curation and Sharing. https://www.caida.org/data/.
[10] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.
[11] Edge-core Wedge 100BF-32X [Online]. https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335, 2019.
[12] ESnet Host and NIC tuning. https://fasterdata.es.net/host-tuning/.
[13] Th Gamer, Chr Mayer, and Marcus Schöller. Pktanon–a generic framework for profile-based traffic anonymization. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 31(2):76–81, 2008.
[14] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the ACM SIGCOMM*, pages 357–371. ACM, 2018.
[15] Martin Izzard. The Programmable Switch Chip Consigns Legacy Fixed-Function Chips to the History Books. https://goo.gl/JKWnQc, September 2016.
[16] Philip Koopman. 32-bit cyclic redundancy codes for internet applications. In *Proceedings International Conference on Dependable Systems and Networks*, pages 459–468. IEEE, 2002.
[17] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
[18] Marc Mendonca, Srini Seetharaman, and Katia Obraczka. A flexible in-network ip anonymization service. In *2012 IEEE international conference on communications (ICC)*, pages 6651–6656. IEEE, 2012.
[19] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM SIGCOMM*, pages 85–98. ACM, 2017.
[20] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The devil and packet trace anonymization. *ACM SIGCOMM CCR*, 36(1):29–38, 2006.
[21] William Wesley Peterson and Daniel T Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.
[22] PFRING: High-speed packet capture, filtering and analysis. https://www.ntop.org/products/packet-capture/pf_ring/.
[23] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.
[24] tcpanon [Online]. http://netweb.ing.unibs.it/~ntw/tools/tcpanon/, 2009.
[25] Tcpreplay sample captures. http://tcpreplay.appneta.com/wiki/captures.html.
[26] TCPurify [Online]. http://irg.cs.ohiou.edu/~eblanton/tcpurify/, 2016.
[27] Barefoot's Tofino. https://www.barefootnetworks.com/technology/.
[28] P4 software switch. https://github.com/p4lang/behavioral-model.
[29] Bapi Vinnakota. P4 with the Netronome Server Networking Platform. https://goo.gl/PKQtC7, May 2016.
[30] Jun Xu, Jinliang Fan, Mostafa H Ammar, and Sue B Moon. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 280–289. IEEE, 2002.
[31] Da Yu, Yibo Zhu, Behnaz Arzani, Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. dshark: a general, easy to program and scalable framework for analyzing in-network packet traces. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, pages 207–220. USENIX Association, 2019.
[32] William Yurcik, Clay Woolam, Greg Hellings, Latifur Khan, and Bhavani Thuraisingham. Scrub-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs. In *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*, pages 49–56. IEEE, 2007.
[33] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM 2015*, volume 45, pages 479–491.