Towards Reproducible Network Traffic Analysis

Jordan Holland
Princeton University
jordanah@princeton.edu

Prateek Mittal
Princeton University
pmittal@princeton.edu

Paul Schmitt
USC/ISI
pschmitt@isi.edu

Nick Feamster
University of Chicago
feamster@uchicago.edu

Abstract
Analysis techniques are critical for gaining insight into network traffic given both the higher proportion of encrypted traffic and increasing data rates. Unfortunately, the domain of network traffic analysis suffers from a lack of standardization, leading to incomparable results and barriers to reproducibility. Unlike other disciplines, no standard dataset format exists, forcing researchers and practitioners to create bespoke analysis pipelines for each individual task. Without standardization, researchers cannot compare “apples-to-apples,” preventing us from knowing with certainty if a new technique represents a methodological advancement or if it simply benefits from a different interpretation of a given dataset.

In this work, we examine irreproducibility that arises from the lack of standardization in network traffic analysis. First, we study the literature, highlighting evidence of irreproducible research based on different interpretations of popular public datasets. Second, we investigate the underlying issues that have lead to the status quo and prevent reproducible research. Third, we outline the standardization requirements that any solution aiming to fix reproducibility issues must address. We then introduce pcapML, an open source system which increases reproducibility of network traffic analysis research by enabling metadata information to be directly encoded into raw traffic captures in a generic manner. Finally, we use the standardization pcapML provides to create the pcapML benchmarks, an open source leaderboard website and repository built to track the progress of network traffic analysis methods.

1 Introduction
Researchers have developed methods to classify network traffic for over 30 years [55]. Classification techniques have been used for a variety of analysis tasks such as application detection, device identification, intrusion detection, and website fingerprinting [13, 43, 46, 61]. In the past, techniques were able to leverage information inside of packet payloads in order to easily traffic classify. However, with the rapid adoption of encrypted network protocols, coupled with ever-increasing data volumes, network traffic analysis has become both more important and more challenging. Research in the field has largely turned to machine learning techniques to address both of these challenges. Yet, after almost 20 years of applying machine learning techniques to various traffic analysis problems, no standard dataset format or comparison methodology exists [51].

The lack of a standardized format had led directly to a reproducibility crisis in traffic analysis research: correctly reproducing previous work is near-impossible. Researchers are required to build bespoke pipelines for each new dataset in which they must engineer a pipeline to parse the format of the dataset, organize the packets according to the task (i.e., applications, intrusions, devices, websites), attach metadata to each set of separated packets, and finally develop and evaluate an analysis technique. Testing even the same technique on a different dataset requires re-inventing the wheel: engineering a new pipeline for a new task. Worse, ambiguous terminology commonly used throughout the field, such as a “traffic flow” or an “application” can lead to researchers, starting from the same dataset, to test analysis techniques on completely different definitions of a task.

These issues lead to multiple downstream problems. First, analysis techniques, such as the performance of a machine learning classifier, are compared against one another when the underlying problem definition differs. Second, the difficulty of creating analysis pipelines has led to a focus on pre-processed datasets, in which the dataset curators release features extracted from the original network traffic as opposed to the raw packets. Techniques stemming from these pre-processed datasets are inherently limited to a subset of the released features and resulting research ultimately ends up being a “model bake-off,” necessarily precluding the discovery of unforeseen features for a given task. Finally, all of these issues create a barrier not only to reproducibility, but to innovation in the field. When compared with the progress of other fields, such as image recognition or natural language processing, techniques in network traffic analysis have been relatively stagnant despite increasing need for accurate analysis methods.

This work examines the state of network traffic analysis research. We first highlight evidence of reproducibility challenges using examples from multiple public datasets. We then highlight underlying causes of these reproducibility issues, including the use of ambiguous terminology, the lack of a standardized dataset format, and a focus on pre-processed datasets. Next, we take our learnings and outline a list of requirements that any proposed standardization solution must meet.

Finally, we introduce pcapML, an open source system that aims to meet our requirements and standardize network traffic analysis research. Rather than focus on a standardized feature set, which previous work has called for, pcapML standardizes network traffic analysis research at the dataset level [12, 18]. pcapML does this by enabling researchers to encode metadata and traffic definitions (i.e., traffic flow directionality) directly into raw traffic captures in a generic manner. Further, pcapML-encoded datasets can still be analyzed by the most popular traffic analysis tools and libraries. Further, we release pcapML-FE, an open source python library which lowers the bar for researchers to incorporate pcapML-encoded datasets into existing pipelines. Lastly, we use pcapML to create the pcapML
2 The Need For Standardization

In this section we examine irreproducibility in the current network traffic analysis ecosystem. First, we investigate literature leveraging several popular network traffic datasets, finding that standard practices lead to incomparable and irreproducible research, visualized in Figure 1. We then survey the literature to outline the practices that lead to irreproducible research.

2.1 Examples of reproducibility challenges.

We begin by examining popular network traffic datasets and literature that tests methods using those datasets. The following subsections highlight three popular datasets examined. We describe how the datasets are curated, the release format of the datasets, and finally compare literature leveraging the datasets to demonstrate examples of irreproducibility in network traffic analysis.

We wish to clearly state that the datasets and papers mentioned are not chosen to point out methodological errors by these specific dataset curators or paper authors. On the contrary, the referenced datasets and papers generally represent a higher standard of reproducibility than many of the datasets and works we examined in our search in that they outline their methodology and dataset usage in such a way that comparisons can be made.

2.1.1 DARPA 1998

Overview. The DARPA 1998 intrusion detection dataset is perhaps the oldest and most well known intrusion detection dataset [43]. The dataset was created in an effort to evaluate and compare intrusion detection methods on the same task. In all, the dataset consists of seven weeks of training data and two weeks of testing data, containing both benign network traffic and 38 attacks comprising 4 broader classes of attack to be classified. Although this dataset has known criticisms in terms of class distribution and experimental collection it has still been widely used due to its long-term availability [52].

Format. The network traffic was captured using tcpdump and released in the PCAP file format [77]. As the challenge consisted of traffic across ten weeks, a PCAP file was released for the traffic captured for each day of the experiment. In all, this results in 45 separate PCAP files. The metadata for the traffic was released in separate “list” files, which are CSV-like in structure. For each PCAP file, a list file was released with labels for TCP and UDP sessions found in the raw traffic. Each record in the metadata files lists the start date, start time in Hour:Minute:Second format, the four-tuple...

![Figure 1: Releasing metadata separate from raw traffic leads to differing versions of the same original task, rendering it impossible to directly compare results.](image-url)
Evidence of Irreproducibility. Researchers using the dataset are required to build a pipeline that parses the raw network traffic by session, associates each record in the metadata file with the set of raw packets that define the session, and extracts information from the session to classify it. To highlight how this methodology can lead to reproducibility and comparability issues in practice, we take the “list” files released with the dataset and calculate the distribution of samples in each of the five classes of traffic. We then examine work from the literature that leveraged the dataset to compare this class distribution with.

Table 1 shows the results of this experiment. We see that authors ultimately experiment with analysis methods on different versions of the dataset. The number of samples in each class of traffic differs both from the originally released dataset and each other. As such, comparisons of the intrusion detection methods in the papers listed and the originally published methods are difficult, if not impossible, to perform. Further, recreating these works is near-impossible as to do so one must recreate an entire bespoke pipeline that lead to a differing version of the underlying dataset. These issues partly occur due to the difficult and messy nature of merging raw network traffic and metadata. Wang et al. describe issues they encounter during this process [86]:

“The traffic format of DARPA1998 is non-split pcap, which must be split into multiple network flow files. In addition, the label files contain a few problems, such as duplicated records and incorrect labels. For example, the label file “Test/Week2/Friday” contains a record of “07/32/1998,” which is an obvious date error. Therefore, the dataset requires preprocessing before the experiments can be conducted.”

2.1.2 CICIDS 2017.

Overview. The CICIDS 2017 intrusion detection dataset was specifically curated to represent a more modern intrusion detection dataset than previously available [67]. The dataset consists of over 40GB of network traffic captured over the course of five days. 15 different classes of network traffic exist in the dataset: 14 types of attack traffic flows and benign traffic flows.

Format. The dataset was released in two formats. First, a pre-processed set of features in which the authors extract features from each labeled flow for use with machine learning methods, using a custom built system. Second, the raw network traffic and associated labels were released as a set of PCAP files, one to two PCAP files per day of the experiment, and a metadata CSV containing information to link the raw traffic flows to a label, such as the flow start timestamps and flow 5-tuples.

Evidence of Irreproducibility. We again process the metadata files released with the dataset, determining the class distribution of the dataset. We then examine the literature for work testing methods on the dataset which reports the class distribution used for their experiments. Table 2 compares the class distribution in the original dataset compared with work leveraging the dataset.

Table 2 again shows that authors ultimately test methods on different interpretations of the same dataset. No row in Table 2 is identical to another. Various works leveraging the dataset merge multiple classes into a single class, subsequently downsampling the number of flows in the merged class. Others ignore classes in the original dataset altogether. Research developing and testing methods on differing interpretations of the same dataset render it difficult to reason about the superiority of any intrusion detection method over another.

2.1.3 VPN-nonVPN

Overview. The third dataset we examine in detail is the VPN-nonVPN dataset, released in 2016 [23]. The dataset consists of traffic from 7 different types of applications, such as “web browsing” or “streaming.” The dataset contains both VPN and non-VPN traffic for each type of application, consisting of 14 different labeled traffic classes.

Format. The dataset is released as a set of PCAP traffic files. Each PCAP file contains traffic generated by an application. Metadata for the traffic is encoded in the name of each PCAP, as each PCAP file is named by the specific application that generated the traffic, such as email11a.pcap. No information mapping the network traffic to the broader application type, which is ultimately used for evaluation in the paper, is released. There is a description of the specific applications that make up each “application type” in the paper itself.

Evidence of Irreproducibility. We survey the literature leveraging this dataset to understand how it is used. Although we are unable to gather detailed information regarding the class distribution for each work, we do find the number of classes of traffic used for evaluation in the original work and subsequent work.

As with previous examples, Table 3 illustrates that works leveraging the dataset interpret the dataset differently from one another. Table 3 also records the authors labeling process as described in each paper, demonstrating how differing versions of the same dataset can be created. Wang et al. even note that they correspond with
The dataset for evaluation is selected from the "ISCX VPN-nonVPN traffic dataset'. As shown in Table 2, the total dataset for evaluation is composed of 15 applications, e.g., Facebook, Youtube, Netflix, etc. The chosen applications are encrypted with various security protocols, including HTTPS, SSL, SSH, and proprietary protocols. A total of 206,688 data packets are included in the selected dataset.

The third dataset focuses on application type classification and is called non-vpn2016. It is obtained by extracting traffic from the non-vpn2015 traffic trace [5, 7, 18, 31]. This dataset contains 25GB raw traffic in the pcap format, which includes 14 network application classes, where 7 for regular traffic (such as Spotify and Facebook) and the rest for the corresponding traffic with VPN-encrypted (such as VPN-Spotify and VPN-Facebook). We relabel the raw traffic into 12 classes to be used in our experiments.

The original authors and still cannot fully reproduce the original dataset, ultimately testing methods on a dataset with two fewer classes than the original work [87].

2.2 Causes of Irreproducibility

Subsection 2.1 presented examples of irreproducibility in network traffic analysis. In this section, we outline and investigate the causes of irreproducibility in network traffic analysis.

2.2.1 Ambiguous Terminology

The datasets examined in Subsection 2.1 describe network traffic and analysis tasks using terminology common throughout the field. The DARPA 1998 analysis task focuses on "sessions", which correspond to individual TCP or UDP "connections" between two IP addresses. The ICIDS 2017 analysis task examines "flows", which they define as bi-directional using the 5-tuple definition of a flow (source IP, destination IP, source port, destination port, protocol). Finally, the VPN-nonVPN dataset focuses on "application" identification, where applications are defined by single traffic flow which is considered to be bi-directional.

Although vernacular is useful in conversation, precise definitions of these terms are required for reproducible research. We perform a simple experiment to better understand the varied definitions of similar terminology in traffic analysis. We examine work appearing in ACM CCS, USENIX Security, PETIS Symposium, IEEE S&P, and NDSS in the years 2021, 2020, and 2019, further supplementing our search with papers already included in Subsection 2.1. We search each paper for two types of terms. First, we look for the most specific term used for defining the analysis task at hand, such as "5-tuple traffic flow" or "application". Second, we examine terms used when defining features for classification, such as "sub-flow" or "tcp-stream". We do not simply report if a term appeared in a paper in any capacity, nor do we consider subsets of terms (i.e., 5-tuple flow does not also include flow). We examine and extract terminology used in 50 papers in total.

Table 3: The current traffic analysis ecosystem creates a burden for dataset curators and researchers using the dataset. Dataset curators are forced to choose a format to release the dataset and work with researchers to rebuild the dataset. Researchers using the dataset must attempt to recreate the datasets faithfully.

| Paper | Citations | Classes | Labeling Description |
|-------|-----------|---------|----------------------|
| Wang et al. [87] | 346 | 12 | Some files such as Facebook_video_tcp can be labeled as either "browser" or "streaming", and all files related to "browser" and "VPN-browser" have this problem. |
| Lotfi et al. [86] | 745 | 12, 17 | The flow features of ISCX dataset have 14 classes of labels, but the raw traffic has no labels, so we labeled pcap files in the dataset according to the description of their paper. |
| Zou et al. [106] | 63 | 12 | The first selected dataset is regenerated from ISCX VPN-nonVPN traffic dataset in order to evaluate the effectiveness of DFR on encrypted traffic classification. ISCX VPN-nonVPN dataset originally has 7 types of regular encrypted traffic and 7 types of protocol encapsulated traffic. Since we mainly focus on evaluating the efficiency on encrypted traffic classification, we will select and label data from these 7 types of regular encrypted traffic, which are all Web browsing, Email, Chat, Streaming, File Transfer, VoIP, and P2P. To be noticed that all other 6 types of encrypted traffic are related to Web browsing, hence we abandoned this class of encrypted traffic referring to Wang's work. |
| Burt et al. [24] | 5 | 12 | The dataset for evaluation is selected from the "ISCX VPN-nonVPN traffic dataset". As shown in Table 2, the total dataset for evaluation is composed of 15 applications, e.g., Facebook, Youtube, Netflix, etc. The chosen applications are encrypted with various security protocols, including HTTPS, SSL, SSH, and proprietary protocols. A total of 206,688 data packets are included in the selected dataset. |

Table 4: Terminology varies among work examined.

| Term               | Referenced By |
|--------------------|---------------|
| 5-tuple flow       | [11, 12, 27, 29, 49, 68, 87, 98–100] |
| 4-tuple flow       | [24, 78, 90] |
| application        | [12, 29] |
| bi-directional flow| [12, 23, 96] |
| bi-directional session | [87] |
| channel            | [10, 53] |
| dns encrypted flow | [70] |
| flow               | [7, 12, 16, 44, 80] |
| network flow       | [43, 56, 84] |
| network trace      | [62] |
| packet trace       | [41] |
| stream             | [31] |
| sub-flow           | [68, 75, 99] |
| tcp connection     | [1, 15, 16, 43, 89] |
| tcp flow           | [33] |
| tcp session        | [4, 43, 88] |
| tcp stream         | [16, 32, 37, 88] |
| tcp sub-flow       | [94] |
| traffic burst      | [97] |
| traffic flow       | [10, 29, 49, 97] |
| traffic stream     | [49] |
| traffic trace      | [7, 31, 64, 71] |
| udp connection     | [43] |
| udp flow           | [80] |
| website trace      | [14, 26, 41, 58, 59, 63, 64, 72, 73] |

Table 4: Terminology varies among work examined.

The flow features of ISCX dataset have 14 classes of labels, but the raw traffic has no labels, so we labeled pcap files in the dataset according to the description of their paper.
Table 5: Dataset formats and separate tasks create a large barrier to testing new methods

| Dataset                      | Citations | Task                  | Raw Traffic (PCAP)? | Metadata Format | Preprocessed Features? |
|------------------------------|-----------|-----------------------|---------------------|-----------------|------------------------|
| DARPA 1998 [43]              | 1,288     | Intrusion Detection   | ☑                   | List File       |                        |
| DARPA 1999 [42]              | 1,250     | Intrusion Detection   | ☑                   | Described On Website |                        |
| UNSW-NB15 [54]               | 1,179     | Intrusion Detection   | ☑                   | ARGUS, BRO, CSV | ☑                      |
| TON_IoT [2]                  | 60        | Cybersecurity Applications | ☑                  | LOG, CSV        | ☑                      |
| Bot-IoT [38]                 | 376       | Botnet Detection      | ☑                   | ARGUS, BRO, CSV | ☑                      |
| CICIDS 2017 [67]             | 1,057     | Intrusion Detection   | ☑                   | CSV             | ☑                      |
| VPN-nonVPN [23]              | 365       | Application Type      | ☑                   | Described In paper | ☑                      |
| Android Malware 2017 [69]    | 938       | Malware Detection     | ☑                   | CSV             | ☑                      |
| Tor 2016 [39]                | 300       | Application Type      | ☑                   | Described On Website | ☑                      |
| CTU-13 [25]                  | 540       | Botnet Detection      | ☑                   | BIARGUS         | ☑                      |
| NSL-KDD [35]                 | 3,339     | Intrusion Detection   | ☑                   | CSV             | ☑                      |
| KDD99 [8]                    | -         | Intrusion Detection   | ☑                   | CSV             | ☑                      |
| Deep Fingerprinting [71]     | 161       | Website Fingerprinting | ☑               | PKL             | ☑                      |

| Dataset                      | Citations | Task                  | Raw Traffic (PCAP)? | Metadata Format | Preprocessed Features? |
|------------------------------|-----------|-----------------------|---------------------|-----------------|------------------------|
| DARPA 1998 [43]              | 1,288     | Intrusion Detection   | ☑                   | List File       |                        |
| DARPA 1999 [42]              | 1,250     | Intrusion Detection   | ☑                   | Described On Website |                        |
| UNSW-NB15 [54]               | 1,179     | Intrusion Detection   | ☑                   | ARGUS, BRO, CSV | ☑                      |
| TON_IoT [2]                  | 60        | Cybersecurity Applications | ☑                  | LOG, CSV        | ☑                      |
| Bot-IoT [38]                 | 376       | Botnet Detection      | ☑                   | ARGUS, BRO, CSV | ☑                      |
| CICIDS 2017 [67]             | 1,057     | Intrusion Detection   | ☑                   | CSV             | ☑                      |
| VPN-nonVPN [23]              | 365       | Application Type      | ☑                   | Described In paper | ☑                      |
| Android Malware 2017 [69]    | 938       | Malware Detection     | ☑                   | CSV             | ☑                      |
| Tor 2016 [39]                | 300       | Application Type      | ☑                   | Described On Website | ☑                      |
| CTU-13 [25]                  | 540       | Botnet Detection      | ☑                   | BIARGUS         | ☑                      |
| NSL-KDD [35]                 | 3,339     | Intrusion Detection   | ☑                   | CSV             | ☑                      |
| KDD99 [8]                    | -         | Intrusion Detection   | ☑                   | CSV             | ☑                      |
| Deep Fingerprinting [71]     | 161       | Website Fingerprinting | ☑               | PKL             | ☑                      |

5-tuple definition (4-tuple and protocol). Worse, each of these definitions can be further divided into uni-directional and bi-directional versions. These distinctions can, and have, lead to the branching methodologies shown in Figure 1. For example, Sharafaldin et al. curated, released, and provided an intrusion detection method for the CICIDS dataset [67]. In their work, they analyze the dataset considering flows as bi-directional 5-tuples, while a 5-tuple flow can also be thought of as unidirectional. Zhang et al. subsequently leverage the raw CICIDS dataset but use software to parse the traffic which considers 5-tuple flows as unidirectional, representing a different task than the original authors. Combining this issue with the varying class distributions shown in Table 2 only creates more methodology branches which are not directly comparable with others.

Even small modifications to the definition of units of traffic can directly affect the results of analysis methods. Draper-Gil et al. examine this, analyzing the performance of models trained on an application identification task using four different flow timeout values, finding that the accuracy of the models varied by up to 3% [23]. In another example, Garcia et al. redefine and re-release a popular public dataset based off of bi-directional traffic flows outperforming a uni-directional flow definition on the same task [25]:

“Each scenario was captured in a pcap file that contains all the packets of the three types of traffic. These pcap files were processed to obtain other type of information, such as NetFlows, WebLogs, etc. The first analysis of the CTU-13 dataset, that was described and published in the paper “An empirical comparison of botnet detection methods” (see Citation below) used unidirectional NetFlows to represent the traffic and to assign the labels. These unidirectional NetFlows should not be used because they were outperformed by our second analysis of the dataset, which used bidirectional NetFlows.”

Applications and Websites. The “application identification” and “website fingerprinting” tasks further exhibit the perils of ambiguous terminology in traffic analysis. Clearly defining the set of packets that are attributable to a single application or website is critical to comparing, contextualizing, and reproducing methods for a given task, as definitions of these tasks can drift over time. In the past, applications often used a single network flow (5-tuple, uni-directional) [13]. However, modern applications commonly leverage multiple simultaneous traffic flows (e.g., Netflix video streaming can include many several underlying flows for a single video session), and recent research has used this information to analyze traffic [19]. Website fingerprinting tasks raise similar questions: does a website comprise of only traffic to and from a single server IP address, or are the multiple connections to other server IPs considered when performing analysis?

2.2.2 Different Tasks, Different Dataset Formats. The three datasets presented in Subsection 2.1 comprise two discrete tasks: application classification and intrusion detection. Yet, each of the three datasets requires a custom-built analysis pipeline to develop and test methods. Although all three datasets store their network traffic in the PCAP capture format, the metadata, which defines how to separate the packets into an “intrusion” or an “application”, is released in a variety of formats. Table 5, which shows the release format of a variety of popular datasets, demonstrates that this is common practice: datasets release network traffic in PCAP format and metadata in a one-off manner. Worse, Table 5 demonstrates that even for researchers working only on a single task (i.e., intrusion detection), there is no common dataset format to interact with. Each dataset format increases the engineering burden on researchers developing new methods, increasing the chances for human error and resulting in branching methodologies shown in Figure 1.

2.3 Preprocessed Datasets Inhibit Innovation.

Another trend seen in Table 5 and subsequent research is a heavy focus on releasing pre-processed traffic as well as calls for standardized feature sets [12, 18]. The highest cited dataset in Table 5,
the NSL-KDD dataset, is a set of pre-processed features derived from the KDD99 dataset which was originally derived from the DARPA1998 and DARPA1999 datasets [35].

The predictive accuracy of any machine learning algorithm, regardless of task, is impacted by the representation of the data that is offered to the model as input. Typically, machine learning pipelines are developed iteratively, observing model performance using different representations of the raw data. It stands to reason that it can be difficult, if not impossible, to know which features should be extracted and how data is best represented for any given task a priori. Publicly released datasets of pre-processed features, rather than raw network traffic that was used to generate the features, prohibit future researchers entirely from exploring new features that the original work may not have considered. This practice directly hinders advances in techniques, forcing research based on pre-processed datasets to ultimately boil down to model “bake-offs.”

3 Standardization Requirements

In this section we outline a list of requirements that a system for standardized traffic analysis must meet. These requirements arise from issues uncovered in Section 2.

Compatible. A large ecosystem of mature and well-known tools and libraries exists for parsing, filtering, and analyzing network traffic, including tcpdump, tshark, wireshark, pf_ring, and libpcap [57, 76, 77, 92, 93]. Any solution that is not inherently compatible with this long-developed ecosystem of tools would amount to re-inventing the wheel and be highly unlikely to be adopted.

Standard. Section 2 explored the pitfalls of requiring researchers to build analysis pipelines for a variety of dataset formats. Any system for standardized traffic analysis must standardize all traffic analysis tasks and datasets under a single dataset format. This requirement significantly lowers the burden on researchers creating and reproducing traffic analysis research, decreasing the chance for human error.

Portable. The standardization solution must unify metadata and raw traffic traces. This requirement eliminates engineering errors that can occur when stitching together metadata and raw network traffic traces. Ultimately, any standardization solution should enable dataset curators to release a single file that contains all raw traffic and associated metadata whenever feasible.

Unambiguous. Section 2 examined the variety of terms that can be used to categorize and identify network traffic, such as traffic flows, traffic sessions, application traffic, and website traces. Vague terminology leads to work that is incomparable. As such, a standardization solution must provide a method for eliminating all ambiguity by encoding this information with the network traffic.

Unprocessed. Pre-processed datasets make the assumption that the pre-processed features best describe the traffic, reducing future methods to only a subset of the preprocessed features. A standardization solution must occur at the lowest granularity possible, the packet level, to avoid any loss of information that could be leveraged in future work.

Reproducible. Any standardization solution must provide users with the ability to uniquely identify all traffic samples in a dataset (i.e., traffic flows, applications, devices). This requirement enhances the reproducibility of traffic analysis work, by enabling researchers to directly compare new methods with previous ones.

4 pcapML

In this section we present pcapML and pcapML-FE (Feature Explorer), open source systems designed to address issues with current practices highlighted in Section 2 while meeting the requirements presented in Section 3. We begin by discussing our solution for the pcapML output file format and embedded metadata. We then present the processing pipeline of pcapML using an example case study. Next, we evaluate costs associated with pcapML using multiple data sets. Finally, we present pcapML-FE, which eases the burden of incorporating pcapML outputs into existing ML pipelines.

4.1 Output Format: PCAPNG

As outlined in Section 3, any standardization solution must meet multiple requirements, including providing a standard format that is compatible with the existing ecosystem of tools.

Almost universally, the networking community has settled on using the PCAP capture format for capturing, storing, analyzing, filtering, and releasing network traffic [28]. The PCAP file format is supported by every popular traffic analysis tool, including tcpdump, libpcap, and wireshark [76, 77, 93]. Unfortunately, the PCAP format, which can be conceptualized as a linked list of raw network packets, provides no generalizable method to unify metadata and raw packets. Techniques to embed metadata information into packet headers, such as optional IP or TCP options, cannot be relied upon as there is no guarantee that these headers will exist in every packet, or that such headers contain the necessary space for metadata. This limitation has led to the status quo for datasets: traffic captures and metadata are stored and released as separate files.

More recently (beginning in 2014), working groups have developed the PCAP next generation (PCAPNG) traffic capture format [47]. While the PCAP format represents a linked list of raw packets, the PCAPNG file format can be conceptualized as a linked list of blocks which encapsulate raw packets. Each packet block contains the raw packet data, the timestamp the packet was captured, and importantly, has a variable length options field. The options field resembles other variable length option fields, such as the TCP options in that it is represented as a list of code, value pairs. pcapML leverages the options field on each packet block to encode metadata directly into the PCAPNG file, unifying metadata and raw network traffic. Our use of PCAPNG satisfies several of our requirements presented in Section 3, as discussed below.

Compatible. The PCAPNG file format has the benefit of already being supported by the most popular libraries and tools to capture and analyze network traffic. Wireshark and tshark are able to both parse and output PCAPNG traffic captures by default, and libpcap and tcpdump can parse PCAPNG captures [77, 91].

Standardized. PCAPNG’s options field provides a means to unify metadata and raw network traffic and create a standard, unified dataset format for holding both network traffic and metadata.
Figure 2: pcapML enables researchers to directly encode metadata into raw traffic captures in a manner that familiar tools used to filter, read, and analyze network traffic can still be leveraged.

There are two options to attach metadata to each packet. First, the PCAPNG file format contains a built-in option type, opt_comment, which allows for arbitrary UTF-8 strings to be attached to every packet in a traffic capture. The advantage of using the built-in option is the guaranteed portability of the encoding. The disadvantage of this method is there is no specified structure for the metadata-encoding as the comment is an arbitrary UTF-8 string. The second option for encoding metadata into the packet block options field is to use a custom option type. This method has the advantage of building structure into the option itself. Unfortunately, the drawbacks include lack of guaranteed portability and flexibility in the future. As such, pcapML leverages the preexisting opt_comment for unifying metadata and raw traffic, using a generic CSV structure inside of the UTF-8 string itself to encode information.

Portable. We leverage the PCAPNG format to directly embed metadata information for every packet in a network traffic trace. As such, pcapML creates a single, portable solution for network traffic datasets, enabling researchers to build analysis pipelines for multiple datasets and tasks around a single dataset format.

Unprocessed. pcapML enables feature exploration by providing a solution at the finest granularity possible, the packet level. Attaching metadata to each individual packet in a dataset incurs no loss of information, allowing researchers to explore, test, and compare potentially unseen features and methods for analyzing the traffic with existing techniques.

4.2 Metadata Format: SampleIDs

As discussed, the PCAPNG file format enables us to directly couple metadata and individual network packets. We now require a generalizable method to match any given packet with its associated metadata and traffic sample. For each traffic analysis task, a traffic sample can be defined differently. In a traffic flow identification task, all packets that belong to a given traffic flow comprise a single traffic sample. For an OS detection task, a single packet may constitute a traffic sample. For anomaly detection, all of the packets inside of a specific time window can comprise a single traffic sample. Finally, for an application identification task, all of the packets created by an application, which may include multiple traffic flows, comprise a single traffic sample.

The variety of goals and traffic sample definitions has led to bespoke pipelines for each individual task. With pcapML, our goal is to find a generalizable solution for the field to build upon. We notice that all traffic analysis tasks have an important common factor: every traffic sample can be defined by a group of one or more packets. We leverage this insight and the ability to encode metadata into PCAPNG traffic captures to create sampleIDs, a simple, generalizable method to identify traffic samples. pcapML’s use of sampleIDs fulfill two requirements that outlined in Section 3.

Unambiguous. pcapML eliminates ambiguity by generating a sampleID for each group of packets that represents a traffic sample for a given analysis task. For example, pcapML can generate a sampleID for every packet belonging to the same traffic flow, application, anomaly, operating system, device, or any other packet grouping. Attaching a sampleID to every packet in a PCAPNG traffic capture, pcapML removes ambiguity surrounding which packets are meant to be associated with a given traffic sample, eliminating all ambiguity arising from vague traffic analysis terms that can be interpreted in multiple ways such as “traffic sessions”, “traffic flows”, and “applications.”

Reproducible. The sampleIDs that pcapML generates provide a means for future researchers to definitively know which traffic samples (i.e., groups of packets) were used for a given analysis task. This capability greatly enhances reproducibility of research, as researchers can know that they are able to work on “apples-to-apples” methodological comparisons at the packet-level. For example, sampleIDs enable researchers to publish the sampleIDs of their training, testing, and validation sets for machine learning based traffic analysis tasks, allowing future researchers to not only ensure they are using the same packets in each traffic sample, but even the same dataset splits. Finally, sampleIDs can increase the depth of understanding across various methods, allowing researchers to uniquely identify the specific samples on which their techniques perform well or poorly.

4.3 Design and Usage

We have described the methods that pcapML uses to satisfy the requirements outlined in Section 3. Figure 2 shows an overview of pcapML, an open source system for standardizing traffic analysis
datasets. In this section outline the capabilities and uses of pcapML in practice using a public dataset to provide illustrative examples.

We leverage the Snowflake Fingerprintability dataset [48] to walkthrough pcapML. The dataset contains over 6,500 DTLS handshakes collected to evaluate the indistinguishability of Snowflake, a pluggable-transport for Tor that leverages WebRTC, with handshakes from other WebRTC applications: Facebook messenger, Google Hangouts, and Discord. As with many publicly released network traffic datasets, the Snowflake dataset was originally released with raw traffic files separated from metadata: a list of PCAP files, one PCAP file for each traffic sample (handshake), and a CSV that maps the traffic in each file to the application that generated the handshake.

Traffic Inputs. We design pcapML to receive and process three types of raw traffic inputs:

- A single PCAP.
- A directory of PCAPs.
- Live traffic.

When processed by pcapML, all input types result in a single, portable output file: a metadata-encoded PCAPNG file. The Snowflake dataset task corresponds to labeling a directory of PCAPs. Listing 1 shows a sample of the Snowflake dataset on disk.

```
$ ls dataset/
ubuntu_chrome_discord_0.pcap
ubuntu_chrome_discord_1.pcap
...
ubuntu_chrome_facebook_0.pcap
ubuntu_chrome_facebook_1.pcap
...
ubuntu_firefox_snowflake_0.pcap
ubuntu_firefox_snowflake_1.pcap
...
ubuntu_firefox_google_0.pcap
ubuntu_firefox_google_1.pcap
...
```

Listing 1: A sample of the DTLS dataset in its originally released format: one PCAP per traffic sample.

Metadata Inputs. pcapML ingests a metadata file along with the raw traffic inputs in order to attach metadata to each traffic sample (i.e., group of packets) in a given dataset. We design pcapML to accept metadata files in pcapML follow a consistent CSV format, with each record containing two or three columns:

- **traffic_filter**, which designates a filter that a set of packets will match and generally represents a single traffic sample.
- **metadata** designates the metadata that will be attached to each packet that matches a specific traffic_filter.
- **group_key**, an optional third column that enables users to generate traffic samples out of multiple traffic_filters by overriding the default sampleID generation and associating all of the packets with the same group_key with a single sampleID.

Traffic Filters. We have implemented three types of traffic filters in pcapML:

- A single PCAP.
- A directory of PCAPs.
- Live traffic.

When processed by pcapML, all input types result in a single, portable output file: a metadata-encoded PCAPNG file. The Snowflake dataset task corresponds to labeling a directory of PCAPs. Listing 1 shows a sample of the Snowflake dataset on disk.

```
$ ls dataset/
ubuntu_chrome_discord_0.pcap
ubuntu_chrome_discord_1.pcap
...
ubuntu_chrome_facebook_0.pcap
ubuntu_chrome_facebook_1.pcap
...
ubuntu_firefox_snowflake_0.pcap
ubuntu_firefox_snowflake_1.pcap
...
ubuntu_firefox_google_0.pcap
ubuntu_firefox_google_1.pcap
...
```

Listing 1: A sample of the DTLS dataset in its originally released format: one PCAP per traffic sample.

Metadata Inputs. pcapML ingests a metadata file along with the raw traffic inputs in order to attach metadata to each traffic sample (i.e., group of packets) in a given dataset. We design pcapML to accept metadata files in pcapML follow a consistent CSV format, with each record containing two or three columns:

- **traffic_filter**, which designates a filter that a set of packets will match and generally represents a single traffic sample.
- **metadata** designates the metadata that will be attached to each packet that matches a specific traffic_filter.
- **group_key**, an optional third column that enables users to generate traffic samples out of multiple traffic_filters by overriding the default sampleID generation and associating all of the packets with the same group_key with a single sampleID.

Traffic Filters. We have implemented three types of traffic filters in pcapML:

- File: file filters are used when running pcapML on a directory of PCAPs and map all the traffic in a single PCAP file to a piece of metadata.
- BPF: BPF filters can be used to filter traffic, where every packet that matches the BPF filter is associated with a piece of metadata.
- Timestamps: timestamp filters map all of the traffic before a given timestamp, after a given timestamp, or between two timestamps to a piece of metadata.

pcapML also has the capability to combine BPF and timestamp filters. An example of the metadata file required to encode the Snowflake dataset is shown in Listing 2. Each filter in the traffic_filter column is simply prepended by the filter type.

```
$ cat metadata.csv
# traffic_filter,metadata,group_key
FILE:dataset/ubuntu_chrome_discord_0.pcap,discord,
FILE:dataset/ubuntu_chrome_discord_1.pcap,discord,
...
FILE:dataset/ubuntu_chrome_facebook_0.pcap,facebook,
FILE:dataset/ubuntu_chrome_facebook_1.pcap,facebook,
...
FILE:dataset/ubuntu_firefox_snowflake_0.pcap,snowflake,
FILE:dataset/ubuntu_firefox_snowflake_1.pcap,snowflake,
...
FILE:dataset/ubuntu_firefox_google_0.pcap,google,
FILE:dataset/ubuntu_firefox_google_1.pcap,google,
...
```

Listing 2: An example metadata file when using pcapml on a directory of pcaps.

pcapML Operation. When run, pcapML first parses the metadata file and generates a vector of traffic filters to be held in memory. For each traffic filter, pcapML generates a unique sequential integer sampleID starting from zero. pcapML originally used a hashing function to generate sampleIDs for each traffic sample, but hashing has drawbacks: shorter output lengths leave the opportunity for hash collisions, while longer output lengths lead to inflated output file size, as a sampleID is attached to each individual packet.

Next, pcapML reads in packets from the given input source, searching the filter vector for matches. pcapML employs two methods for searching for a filter match: when matching FILE filters (while processing a directory of PCAPs), pcapML loads in the sampleID and metadata associated with all of the packets in the file before processing in order to avoid vector lookup overhead for each packet. Conversely, when matching BPF or timestamp filters, pcapML linearly searches the filter vector until it finds a match. We implemented a small optimization in the vector search by beginning the search from the last matched filter, rather than searching the vector from the beginning for all packets. This optimization takes advantage of the bursty nature of traffic from hosts or in flows, allowing for quick lookups for adjacent packets that match identical filters without adding significant system complexity.

Finally, pcapML outputs each packet that matches a traffic filter to the designated output file, leveraging the PCAPNG packet block options to encode the sampleID and metadata associated with the packet in a sampleID,metadata UTF string. Listing 3 shows that we can encode the Snowflake Fingerprintability dataset with metadata using a single command.
Listing 3: Encoding a dataset with metadata using pcapml.

```
$ pcapml -O dataset/ -L metadata.csv -W dataset.pcapng
```

**pcapML Output.** pcapML outputs PCAPNG traffic capture files that directly couple metadata and raw network traffic while meeting the portability and compatibility requirements. The result of the above command can still be parsed by popular tools such as tcpdump and tshark as shown in Listing 4.

```
$ pcapml -D dataset/ -L metadata.csv -W dataset.pcapng
```

Listing 4: pcapml-encoded files are portable to other tools.

Using tools with advanced PCAPNG functionality, such as tshark, we can easily inspect the metadata encoded into the traffic capture. As shown in Listing 5, we see the sampleID for each packet is associated with (in this case, a DTLS handshake), followed by the label for the traffic sample.

```
$ tshark -r dataset.pcapng
```

Listing 5: tools such as tshark can be used to inspect or extract pcapML encoded datasets.

**Sorting by SampleID.** By default, pcapML outputs packets in the order they are processed. When processing a directory of PCAPs using file filters, the packets are naturally sorted by the sampleID as the traffic samples are already split before being ingested. However, when attaching metadata to a single PCAP or to live traffic, the output PCAPNG is in timeseries order by default. As it is often beneficial to sort the packets first by sampleID, and then by time order; pcapML provides this functionality, shown in Listing 6.

```
$ tcpdump -r dataset.pcapng
```

Listing 6: Sorting a pcapML encoded dataset.

**Backwards Compatibility.** Although pcapML-encoded PCAPNGs are portable across many different tools and tasks, not every traffic analysis system supports the PCAPNG traffic capture format. As such, we have implemented functionality in pcapML to revert pcapML-encoded PCAPNGs out to PCAPs, using one PCAP file per traffic sample.

```
$ pcapml -M unsorted_dataset.pcapng -s -W sorted_dataset.pcapng
```

In this section we evaluate the performance of pcapML in a variety of dimensions, including the time it takes to label a variety of datasets, the cost (in disk space) of attaching information to every packet in a traffic capture, and the speed at which the public implementation of pcapML can label live traffic.

**pcapML Metadata Overhead.** We first examine the overhead file size cost of pcapML by setting up an experiment where we increase the size of the metadata attached to each packet in a given traffic capture. For this experiment, we leverage tcpreplay to replay traffic and use the bigFlow traffic capture to provide a traffic example [5,6]. The traffic capture contains almost 800,000 packets across over 40,000 traffic flows and 132 applications with an average packet size of 449 bytes.

We select 20 random source IP addresses (out of a possible 3,218 IP addresses in the capture) from the capture to tag with metadata. We use the BPF filtering mechanism in pcapML to tag all of the traffic originating from the selected IP addresses, attaching a static number of metadata bytes to each packet matching our filters. Finally, we strip the resulting pcapML-encoded PCAPNG of the metadata, transforming it into a PCAP in order to calculate the disk overhead due to the metadata. We run the experiment 10 times for each metadata size tested.

Table 6 shows the results of this experiment. As expected, we see that the disk overhead of pcapML increases with the size of the metadata.

| Metadata Size (Bytes) | Mean | Min | Max |
|-----------------------|------|-----|-----|
| 1                     | 8.7  | 5.1 | 12.3|
| 5                     | 8.1  | 3.8 | 13.3|
| 10                    | 10.3 | 4.6 | 19.1|
| 20                    | 12.0 | 4.6 | 24.2|
| 40                    | 16.3 | 6.5 | 29.4|
| 80                    | 19.3 | 11.1| 27.8|
| 160                   | 30.3 | 17.1| 51.1|

Table 6: The disk overhead of pcapML encoded traffic captures generally increase with the size of the metadata.

Ultimately, a number of factors influence the size of the pcapML-encoded PCAPNG file when compared to a PCAP without metadata. First, the overhead of pcapML can increase due to a large number of samples in a dataset. The number of bytes required to write higher sampleID values to file increases with the number of samples to be encoded. Second, the distribution of packet sizes can influence the overhead cost of pcapML. A small number of large packets will incur relatively lower overhead than a large number of smaller packets with a smaller average size.
Table 7: pcapML can encode datasets in an efficient amount of time with overhead cost depending on the amount of metadata to be attached.

| Dataset          | # Pcap Files | Disk Size | PcapML-Encoded Size | Stripped Pcap Size | % Increase Over Raw PCAP | Time To Encode (Seconds) | Time To Split (Seconds) |
|------------------|--------------|-----------|---------------------|--------------------|-------------------------|--------------------------|-------------------------|
| Active Case Study| 274,009      | 1.1 GB    | 30 MB               | 25 MB              | 12                      | 3                        | 7                       |
| Application Case Study | 5,787 | 33 MB    | 20 MB               | 18 MB              | 10                      | 0.2                      | 0.4                     |
| Cross Market Case Study | 521   | 8.6 GB   | 8.8 GB              | 8.4 GB             | 5                       | 12                       | 21                      |
| netML IDS        | 558,884      | 12 GB     | 8.0 GB              | 7.4 GB             | 8                       | 17                       | 32                      |
| netML IoT        | 149,446      | 16 GB     | 13 GB               | 12 GB              | 8                       | 39                       | 46                      |
| OS Case Study     | 124,390      | 529 MB    | 241 MB              | 187 MB             | 22                      | 2                        | 3                       |
| Video Case Study  | 20,980       | 330 MB    | 414 MB              | 288 MB             | 30                      | 2                        | 5                       |

Figure 3: pcapML can encode metadata into live traffic in configurations that satisfy many experimental use cases.

experimental use cases.

interface and uses a custom PCAPNG writer for matching and writing packets to disk. The existing implementation of pcapML uses a single thread.1

We again leverage tcpreplay, and the same bigFlows.pcap traffic capture as our previous experiment examining disk overhead. We vary two input parameters. First, we vary the number of samples that pcapML is attempting to tag. Recall that, for each packet, pcapML must check each traffic filter in the vector until it finds a match. As such, the time pcapML takes to check for a match increases with the number of filters in the search vector. Next, we vary the speed at which packets are replayed on the interface to understand the processing capabilities of pcapML and at which rates losses occur.

We run each individual (number samples, speed) configuration five times and report the recorded loss of each combination. Figure 3 examines the results of the experiment in detail, showing the loss for each configuration tested. We see that pcapML can generally filter and tag live traffic at 1 Gbps with no loss for up to 256 defined traffic samples. pcapML can tag up to 100 samples with zero loss, but struggles with larger numbers of samples. Finally, pcapML can

We anticipate that pcapML could be tuned and refactored for high performance network environments using optimized packet capture libraries and concurrency. We leave this for future work.

1We anticipate that pcapML could be tuned and refactored for high performance network environments using optimized packet capture libraries and concurrency. We leave this for future work.

Table 7 demonstrates the overhead cost across the eight encoded datasets by stripping the pcapML-encoded PCAPNG of the attached metadata and transforming it into a PCAP traffic capture. The cost of attaching the sampleID and metadata to each of these datasets ranges from 5 to 30%, depending on the factors previously discussed. Finally, Table 7 reports the amount of time pcapML took to encode the dataset and decode the dataset for backwards compatibility. We see that even the largest dataset, containing 18GB of traffic, takes less than a minute to encode and decode using pcapML.

pcapML Performance on Live Traffic. pcapML can capture and attach metadata to traffic on a live network interface. This functionality can be useful for researchers who wish to run traffic experiments as they can capture, label, and ID the traffic samples for their experiment in real time. We measure the speeds which pcapML can capture and attach metadata to incoming packets while avoiding packet losses. pcapML leverages l1bpcap to read packets from an

sized packets. Finally, the size of the metadata to be attached to the packet will influence the overhead cost of pcapML. As the metadata is attached to each packet with the associated sampleID, larger metadata can increase disk space overhead.

pcapML Overhead On Public Datasets. Next, we evaluate pcapML on eight public datasets. These eight datasets were published by Holland et al [29] and represent both 1) a variety of tasks: including device identification, application identification, intrusion detection, and OS detection; and 2) a variety of traffic types: containing various types of packets from DTLS traffic handshakes to UDP and TCP traffic.

Table 7 examines the overheads and performance of using pcapML on these eight datasets. First, we see that each dataset was originally released as a set of PCAPs, one PCAP per traffic sample. Table 7 includes both the number of files in the originally formatted dataset and the size of the dataset on disk as reported by du. Next, we see the size of the pcapML-encoded PCAPNG file after processing the files using pcapML. In many cases, the size of the dataset decreases significantly—this is a product of reducing the number of files on disk, as filesystem block size overheads dominate datasets with large numbers of files. In other cases, the size of the dataset increases on disk when attaching metadata using pcapML.

pcapML can capture and attach metadata to incoming packets while avoiding packet losses, pcapML leverages l1bpcap to read packets from an

We again leverage tcpreplay, and the same bigFlows.pcap traffic capture as our previous experiment examining disk overhead. We vary two input parameters. First, we vary the number of samples that pcapML is attempting to tag. Recall that, for each packet, pcapML must check each traffic filter in the vector until it finds a match. As such, the time pcapML takes to check for a match increases with the number of filters in the search vector. Next, we vary the speed at which packets are replayed on the interface to understand the processing capabilities of pcapML and at which rates losses occur.

We run each individual (number samples, speed) configuration five times and report the recorded loss of each combination. Figure 3 examines the results of the experiment in detail, showing the loss for each configuration tested. We see that pcapML can generally filter and tag live traffic at 1 Gbps with no loss for up to 256 defined traffic samples. pcapML can tag up to 100 samples with zero loss, but struggles with larger numbers of samples. Finally, pcapML can

We anticipate that pcapML could be tuned and refactored for high performance network environments using optimized packet capture libraries and concurrency. We leave this for future work.
The ultimate goal of many traffic analysis tasks is to extract information from a set of packets to identify sets of traffic samples, such as identifying different devices on a network. A non-negligible portion of traffic analysis work is done using Python, along with several popular libraries. To facilitate exploration of new analysis methods and incorporation of pcapML into existing pipelines, we have developed pcapML-FE, an open source tool which leverages the standardized dataset format of pcapML to enable researchers to focus on analysis, rather than dataset formatting.

pcapML-FE is a Python module that interfaces directly with pcapML encoded datasets, exposing an iterator over individual traffic samples and their associated metadata. pcapML-FE can transform packets into popular Python packet analysis libraries such as scapy and dpkt \([66, 74]\) in a single line of code. Finally, we point out that one of the most important capabilities that pcapML-FE and pcapML provides is simplifying the search for generalizable traffic analysis methods. The standardized dataset format of pcapML and standard interface of pcapML-FE enables researchers to test methods across datasets and tasks by simply loading in a new dataset. Below is an example of pcapML-FE loading and iterating over the Snowflake Fingerprintability dataset encoded with pcapML.

```python
import argparse
from pcapml_fe_helpers import *

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('pcapml_dataset')
    args = parser.parse_args()

    for traffic_sample in pcapml_fe.sampler(args, pcapml_dataset):
        analysis_method(traffic_sample)

def analysis_method(sample):
    print(sample.sid)
    print(sample.metadata)

for pkt in traffic_sample.packets:
    # Raw bytes and timestamp
    print(pkt.ts, pkt.raw_bytes)
    # Transform to Scapy packet in one line
    spacket = scapy_readEther(pkt.raw_bytes)
```

Listing 7: pcapML-FE enables researchers to focus on analysis methods. Testing an analysis method on a different dataset only requires loading in a different pcapML encoded dataset.

5 pcapML Benchmarks

pcapML represents a "narrow waist" in network traffic analysis, providing a standardization solution for storing and processing network traffic datasets. Such a narrow waist can be used for broader impact by leveraging standardization to centralize and track progress in the field. Along with pcapML, we have created the pcapML benchmarks \(^2\), an open source repository and public leaderboard with the goal of centralizing and standardizing network traffic analysis research. As of writing, the pcapML benchmarks contain seven datasets across five discrete tasks, including website fingerprinting, device identification, malware detection, and application identification.

Any researcher or practitioner can add a new task, a new dataset, or submit results of a method on one of the public leaderboards. The only requirement being that each submitted task and dataset be encoded with pcapML for standardization. For dataset curators, the pcapML benchmarks provide a central repository to list their datasets, tasks, and any initial results on the dataset, without having to develop custom methods for storing and releasing each dataset. For dataset users, the pcapML benchmarks ensure any methods developed and evaluated can be directly compared with any other technique using the same pcapML benchmark. Further, a central repository of standardized datasets enables dataset users to easily test methods on a variety of traffic analysis tasks, encouraging researchers to create and evaluate generalizable methods. Finally, for the field, a centralized benchmark repository provides an avenue for tracking the progress of techniques, making it easier to discern when a methodological breakthrough has occurred.

6 General Recommendations

We have highlighted issues that simultaneously inhibit reproducibility and innovation in network traffic analysis. While pcapML is an initial solution for reproducible network traffic analysis, it is not a panacea. Moving forward, we urge the community to consider approaching this research area with a broader lens. Here we discuss the different community stakeholders and their role in making network traffic analysis research more reproducible, and subsequently more impactful.

Dataset Creators. Dataset creators curate and release network traffic datasets for the community to use, inherently guiding downstream research. The community must converge on a standard dataset release format to enable sound research and lower the barrier for new techniques to be developed. We have introduced pcapML as a possible avenue for generating standard network traffic datasets, but any format which meets the requirements outlined in Section 3 would improve upon the current status quo.

Dataset Users. Dataset users play an equally important role in improving reproducibility in network traffic analysis. Many traffic analysis techniques report high levels of performance on a given task (e.g., \(\geq 90\) precision) such as website fingerprinting or intrusion detection. As a field, we are entering an era of diminishing returns as we near the point in which many existing methods and techniques perform "well enough" for most tasks. New methods
are likely to represent relatively small, yet vitally important performance increases. For instance, even performance increases of 1\% can denote a methodological advancement. However, without exact comparisons of methods, such small performance increases can be difficult to ascribe to a method versus other factors. A set of best practices when performing analysis will enable the field to better understand when a methodological advancement has occurred.

First, we must focus efforts on testing methods on full datasets when possible. Due to the large number of samples found in many network traffic datasets, it has become accepted to evaluate methods on sampled versions of full network traffic datasets. This practice can be dangerous due to the temporal nature of network traffic (i.e., events are not necessarily independent). If a dataset is sampled to evaluate a method, either the sampled dataset must be re-encoded and released, or in the case of pcapML, the sampleIDs used in evaluation can easily be released. If machine learning techniques are used for a task, releasing specific training and testing datasets (or sampleIDs) will help to directly compare methods. Third, if a dataset is leveraged for a purpose other than its original release, such as re-labeling the traffic for a different task, the dataset should be re-encoded and released to guarantee reproducible research. Finally, when releasing pre-processed data, such as features used to train a model, a direct link to the raw network traffic should be traceable. For example, releasing a CSV of features which includes a sampleID column will enable future work to better contextualize new methods with old ones.

7 Related Work

This section explores recent standardization and reproducibility work as well as work related to pcapML.

PCAPNG. The PCAPNG traffic capture format has been in development since 2014. As such, research has leveraged the format in a variety of ways. Le et al. leverage the PCAPNG file format in AntMonitor, a system for monitoring mobile devices, to attach mobile application names to raw packets [40]. Velea et al. also leverage the PCAPNG traffic capture format to encode pre-processed feature information, such as the use of encryption, the protocol, and the number of packets in a flow using a custom-developed block option [82, 83]. In contrast to these works, our work focuses on building a generalizable system for network traffic analysis tasks by encoding arbitrary metadata onto packets.

Reproducibility and ML. A trove of recent research has examined and highlighted irreproducibility across a variety of fields. Most famously, the Open Science Collaboration highlighted the reproducibility crisis in psychological science [21]. More recently, much work has focused on the reproducibility of applied machine learning research. Applied machine learning reproducibility failures have been brought to light in fields including: neuroimaging, bioinformatics, medicine, software engineering, toxicology, radiology, epidemiology, political science, and nutrition [3, 17, 20, 22, 30, 34, 45, 50, 60, 65, 79, 81, 90, 95].

Most related to our work is Arp et al.’s examination of applied machine learning in a variety of computer security research [9]. Arp et al. examine the use of incorrect application of machine learning techniques to a single intrusion detection dataset, demonstrating the methods used in Mirsky et al. are likely overly complex for the task [53]. In contrast, our work examines the area of network traffic analysis more generally, demonstrating that any analysis performed, not only machine learning, is irreproducible due to issues existing before the analysis stage of a pipeline.

8 Conclusion

In this work we highlighted a reproducibility crisis facing network traffic analysis research. We examined the usage of multiple popular datasets, highlighting how the existing ecosystem leads to researchers testing methods on different versions of the same dataset. We then inspected the literature to outline the barriers to reproducibility in the field, which we used to develop a list of standardization requirements.

Given our findings, we introduced pcapML, an open source system which meets the outlined requirements. pcapML standardizes dataset curation and usage by enabling researchers to directly encode metadata into raw traffic traces, eliminating ambiguous language and engineering errors that stem from the variety of dataset formats that currently exist. We evaluate pcapML across multiple dimensions, including the increase in dataset size when encoding traffic datasets using pcapML and pcapML’s capability to encode metadata into traffic in real-time. Finally, we demonstrate the broader impact that standardization can have on the field by creating the pcapML benchmarks, a public leaderboard website and repository built to track the progress of methods in network traffic analysis. Ultimately, we see pcapML and the pcapML benchmarks as an avenue to propel the field forward, enabling rapid innovation through centralization and standardization.

References

[1] Geoffrey Alexander, Antonio M Espinoza, and Jedidiah R Crandall. 2019. Detecting TCP/IP Connections via IPID Hash Collisions. Proceedings on Privacy Enhancing Technologies 2019, 4 (2019).
[2] Abdullah Alsaedi, Nour Mosattafa, Zahir Tari, Abdul Mahmood, and Adnan Anwar. 2020. TON, IoT telemetry dataset: A new generation dataset of IoT and IoT for data-driven intrusion detection systems. IEEE Access 8 (2020), 165130–165150.
[3] Vinicius M Alves, Joyce Borba, Stephen J Capuzzi, Eugene Muratov, Carolina H Andrade, Ivan Rusyn, and Alexander Trophsa. 2019. Oy vey! A comment on “Machine learning of toxicological big data enables read-across structure activity relationships outperforming animal test reproducibility”. Toxicological Sciences 167, 1 (2019), 3–4.
[4] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. 2019. Nearby Threats: Reversing, Analyzing, and Attacking Google’s Nearby Connections on Android. (2019).
[5] AppNeta. [n.d.]. Sample Captures. https://tcpreplay.appneta.com/wiki/captures.html
[6] AppNeta. [n.d.]. Tcpreplay - Pcap editing and replaying utilities. https://tcpreplay.appneta.com/
[7] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. 2018. Keeping the smart home private with smart (er) iot traffic shaping. arXiv preprint arXiv:1802.00953 (2018).
[8] The UCI KDD Archive. [n.d.]. KDD Cup 1999 Data. https://kdd.ics.uci.edu/ databases/kddcup99/kddcup99.html
[9] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2020. Dos and Don’ts of Machine Learning in Computer Security. arXiv preprint arXiv:2010.09470 (2020).
[10] Diogo Barradas, Nuno Santos, and Luis Rodrigues. 2018. Effective detection of multimedia protocol tunneling using machine learning. In 27th {USENIX} Security Symposium ({USENIX} Security 18). 169–185.
[11] Diogo Barradas, Nuno Santos, Luis Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. 2021. FlowLenses: Enabling Efficient Flow Classification for ML-based Network Security Applications. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium.
