Traffic Generation using Containerization for Machine Learning

Henry Clausen
henry.clausen@ed.ac.uk
University of Edinburgh
Edinburgh, UK

Robert Flood
s1784464@ed.ac.uk
University of Edinburgh
Edinburgh, UK

David Aspinall
University of Edinburgh
Edinburgh, UK
The Alan Turing Institute
London, UK

ABSTRACT

KEYWORDS

Network security, datasets, machine learning, intrusion detection

ACM Reference Format:
Henry Clausen, Robert Flood, and David Aspinall. 2019. Traffic Generation using Containerization for Machine Learning. In *DYNAMICS '19: DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security Workshop, December 09–10, 2019, San Juan, PR.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

.....
.....
.....
This work provides the following contributions:

(1) We present a novel network traffic generation framework that is designed to improve several shortcomings of current datasets for NIDS evaluation. This framework is openly accessible for researchers and allows for straightforward customization.
(2) We define four new requirements a network intrusion dataset should fulfil in order to be suitable to train machine-learning based intrusion detection methods.
(3) We perform a number of experiments to demonstrate the suitability and utility of our framework.

1.1 Outline

The remainder of the paper is organized as follows. Section 2 discusses existing NIDS datasets and the problems that arise during their usage as well as background information about network traffic data formats and virtualization methods. The section concludes with a set of requirements we propose to improve the training and evaluation of machine-learning-based methods. Section 3 describes the general design of our framework, and how it improves on the discussed problems in existing datasets. We also discuss a specific example in detail. Section 4 discusses several experiments to validate the improvements and utility our framework provides. Section 5 concludes the results and discusses limitations of our work and directions for future work.

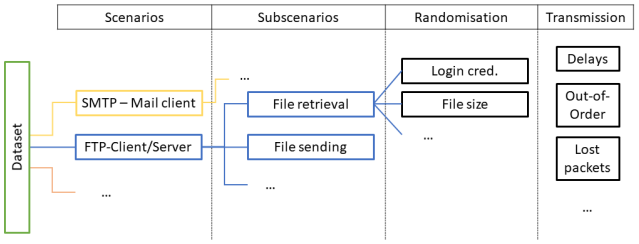

Figure 1: Visualization of the different levels at which traffic variation is introduced in DetGen.

2 BACKGROUND

2.1 Data formats

2.2 Related work and existing datasets

2.3 Problems in modern datasets

2.4 Containerization with Docker

2.5 MiniNet

3 DATASET REQUIREMENTS

Variation.

Ground truth.

Modularity.

Scalability.

4 DESIGN

4.1 Modes of Operation

To cover a range of activities, the containers in our framework are arranged in different configurations corresponding to particular *capture scenarios*. Running a given capture scenario triggers the launch of several Docker containers, each with a scripted task specific to that capture scenario. A simple exemplary capture scenario may consist of a containerized client pinging a containerized server. We ensure that each Docker container involved in producing or receiving traffic will be partnered with a `tcpdump` container, allowing us to collect the resulting network traffic from each container's perspective automatically.

We outline different stages within the creation of a dataset at which traffic variation is introduced. Figure 3 visualizes this process.

DYNAMICS '19, December 09–10, 2019, San Juan, PR

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

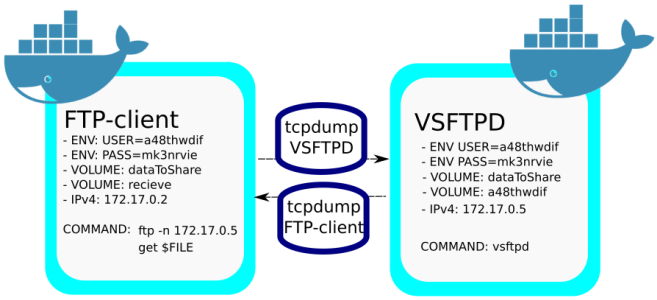

Figure 2: Diagram of FTP scenario

4.2 Scenarios and subscenarios

4.3 Randomization

4.4 Network transmission

4.5 Implementation Process

4.6 Simple Example Scenario - FTP server

We review the design of a prototypical capture scenario, namely, an FTP server and client interaction. The interaction is initiated by a single script, which allows the user to specify the length of the interaction, the number of times the interaction takes place as well as the specific subscenario. The script generates a random ftp username and password, creating the necessary *User* directory on the host machine before calling the Docker-compose file which creates a bridge network. Subsequently, the necessary containers are then started which, in this case, consist of a VSFTPD server, a client with ftp instaled and two containers running tcpdump to capture all of the traffic emitted and received by the client and server respectively into separate `.pcap`-files. These `.pcap`-files are shared with the host machine via a shared volume. The host machine also shares:

- A *dataToShare* volume containing files that can be downloaded by the client.
- The *User* directory with the server, which contains the same files as the *dataToShare* folder.
- An empty *receive* folder with the client into which the files will be downloaded.
- The random username and password is shared with the client container so it can authenticate itself to the server.

Up to this point, no network traffic has been generated and the containers are now ready to begin communicating with one another. For this particular interaction between an FTP server and client, we want to ensure that it is possible to capture the many ways in which an FTP session may develop. For instance, the client may seek to download files via the `get` command or the `put` command, alongside many other possibilities. We define 13 possible capture subscenarios intended to encapsulate a wide range of potential FTP sessions. These include downloading a single file using `get` or `put`, downloading every file using `mget` or `mput`, deleting every file from the server and requesting files from the server without the necessary authentication.

After the scenario ends, both the *User* directory and any downloaded files are removed from the host machine. The containers are then stopped and the bridge network is torn down. All necessary containers, volumes and scripts are in the same position prior to initiating the scenario — barring any generated `.pcap`-files — allowing for the scenarios to be started repeatedly with minimal human interaction. The `.pcap`-files are tagged with information about the time of creation, executed scenario and subscenario, and the container generating the traffic.

4.7 Dataset creation

Our framework generates network datasets consisting of a single interaction, but it is possible to coalesce these datasets to create larger datasets with a wide variety of traffic, albeit with some caveats. Due to the networking constraints of the Docker virtual network, such as limitations regarding clashing ports, running many of our Docker scenarios simultaneously over a large period of time is unfeasible. Thus, to ensure that the generated traffic is suitably heterogeneous, numerous datasets must be generated before being coalesced into a main dataset. If done naively, this presents a problem. As discussed by Shiravi et al. [**?**], merging distinct network data in an overlapping manner can introduce inconsistencies. For instance, if one wanted to create a dataset containing both normal webserver traffic and traffic originating from a Denial of Service attack, it would not work to generate these two datasets separately before merging them together. If these two events really did occur simultaneously, the high network throughput of the latter would likely effect the packet timings of the former.

To avoid such inconsistencies, we create larger datasets by collecting data in consecutive chunks of fixed time. Within each chunk, several scenarios are run simultaneously. All `.pcap`-files collected during a given chunk can be merged together. It is then simple to stitch together all of these chunks into a single `.pcap`-file using a combination of `Mergecap` [**?**] and `Editcap` [**?**]. This allows us to shift the timings of each `.pcap`-file by a fixed amount such that all of our chunks occur in succession whilst maintaining the internal consistency of each chunk.

4.8 Scenarios

Our framework contains 29 scenarios, each simulating a different benign or malicious interaction. The protocols underlying benign scenarios were chosen based on their prevalence in existing network traffic datasets.These datasets consist of common internet protocols such as HTTP, SSL, DNS, and SSH. According to our evaluation, our scenarios can generate datasets containing the protocols that make up at least 87.8% (MAWI), 98.3% (CIC-IDS 2017), 65.6% (UNSW NB15), and 94.5% (ISCX Botnet) of network flows in the respective dataset. Our evaluation shows that some protocols that make up a substantial amount of real-world traffic are glaringly omitted by current synthetic datasets, such as BitTorrent or video streaming protocols, which we decided to include.

In total, we produced 17 benign scenarios, each related to a specific protocol or application. Further scenarios can be added in the future, and we do not claim that the current list exhaustive. Most of these benign scenarios also contain many subscenarios where applicable.

Name	Description	#Ssc.
Ping	Client pinging DNS server	1
Nginx	Client accessing Nginx server	2
Apache	Client accessing Apache server	2
SSH	Client communicating with SSHD server	5
VSFTPD	Client communicating with VSFTPD server	12
Scrapy	Client scraping website	1
Wordpress	Client accessing Wordpress site	1
Syncthing	Clients synchronize files via Syncthing	1
mailx	Mailx instance sending emails over SMTP	2
IRC	Clients communicate via IRCd	2
BitTorrent	Download and seed torrents	3
SQL	Apache with MySQL	2
NTP	NTP client	2
Mopidy	Music Streaming	5
RTMP	Video Streaming Server	1
WAN Wget	Download websites	5
SSH B.force	Bruteforcing a password over SSH	3
URL Fuzz	Bruteforcing URL	1
Basic B.force	Bruteforcing Basic Authentication	2
Goldeneye	DoS attack on Web Server	1
Slowhttptest	DoS attack on Web Server	4
Mirai	Mirai botnet DDoS	3
Heartbleed	Heartbleed exploit	1
Ares	Backdoored Server	3
Cryptojacking	Cryptomining malware	1
XXE	External XML Entity	3
SQLi	SQL injection attack	2
Stepstone	Relayed traffic using SSH-tunnels	2

Table 1: Currently implemented traffic scenarios along with the number of implemented subscenarios

The remaining 12 scenarios generate traffic caused by malicious behavior. These scenarios cover a wide variety of major attack classes including DoS, Botnet, Bruteforcing, Data Exfiltration, Web Attacks, Remote Code Execution, Stepping Stones, and Cryptojacking. Scenarios such as stepping stone behavior or Cryptojacking previously had no available datasets for study despite need from academic and industrial researchers.

We provide a complete list of implemented scenarios in Table 1.

5 VALIDATION EXPERIMENTS

A framework that generates network traffic does not necessarily provide realistic and useful data. To evaluate the utility of our Docker framework, we construct a series of experiments. We have two goals in mind. First, we want to demonstrate that the traffic generated is sufficiently representative of real-world traffic. Second, we want to demonstrate that having a framework to continually generate data compared to static datasets benefits evaluating the efficacy of intrusion detection systems.

The first experiment provides a general verification of the reproducability of our framework, which is required for guarantee the ground truth of the produced data. The second experiment demonstrates that the WAN-characteristics we emulate for our data make it quasi non-distinguishable from real WAN traffic. Our third experiment then demonstrates the advantage of unlimited data generation capabilities for training ML-based traffic classification.

5.1 Reproducible scenarios

To provide ground truth, we have to guarantee that our implemented scenarios and subscenarios are consistent and reproducible upon repeated execution. This applies both to consistency for external influences on the host, such as increased computational load, as well as internal consistency of the implemented script execution.

It is impossible to guarantee that each scenario will produce a truly 'deterministic', or repeatable, output due to differences in network conditions, computational times, or input. Instead, we aim for our data to be *reproducible up to networking and computational differences*. This means that when running a scenario multiple times, we expect the quantities of most packets to be largely identical. We do expect some packets to exhibit greater variation due to non-determinism in the underlying protocols, Fig. 5 outlines this behavior in terms of interarrival times and packet sizes.

To measure how consistent our scenarios are, we generate 500 `.pcap` files for three different implemented scenarios, namely the Apache, the VSFTPD, and the SSH scenario. These were generated consecutively under different host CPU load. We did not apply any delays or other NetEm traffic controls.

We assess the consistency of a scenario across different `.pcap` files by comparing all generated `.pcap` files pairwize. We measure this by the similarity of the connections captured.

To test the similarity of two connections, we extract the sample distributions of the packet interarrival times and packet sizes overall, upstream and downstream. We define two connections as similar if the two distributions for each of these quantities pass an equality test. We use the two-sample Kolmogorov-Smirnov (K-S) test, a non-parametric statistical test for the equality of two continuous one-dimensional distributions [**?**], with a p-value of 0.01.

As all tested files passed this similarity test, we conclude that these scenarios yield consistent and reproducible results. As other scenarios follow the same setup and launch commands, we expect the results to stay the same as long as the involved containers are consistent in their behavior.

5.2 Explorating Artificial Delays

Most traffic our framework generates is transported over Docker's virtual network and therefore does not succumb to problems associated with normal network congestion, such as packet loss, corruption and packets arriving out of order. A realistic dataset should include these phenomena, which is why we developed wrapping scripts that allow us to artificially add delays as well as packet loss and corruption, using NetEm. Choosing the parameters is not straightforward; it is not clear how close to real-world traffic such

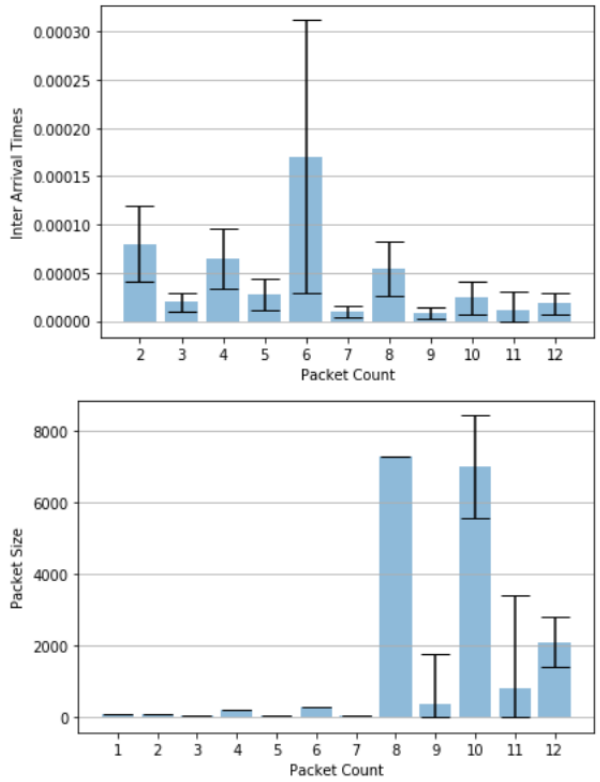

Figure 3: Means of IATs & packet sizes along with standard deviation bars for the first twelve packets in the Apache scenario.

network emulation techniques are. This is especially true for packet delays, which are described by continuous distributions and often have temporal correlation.

Furthermore, the high effective bandwidth of the Docker virtual network resulted in traffic with extremely short inter-arrival times (IATs, defined as the time between two packet arrivals). Therefore, we devote considerable time to demonstrating that it is possible for traffic generated by our Docker framework to conform to real-world IAT distributions when altered using NetEm.

Datasets. We create two classes of datasets, one which is representative of 'real-world' traffic, and one which has been generated from our Docker framework. For simplicity, we only consider datasets consisting of FTP traffic.

For the real-world dataset, we set up a containerized VSFTPD server running on a *Google Compute* virtual machine located in the Eastern United States, and a containerized FTP client on our local host. We then ran a series of our scripted interactions between the two machines, generating 834 megabytes of data in 250964 packets. These interactions consisted of several FTP commands with various network footprints. We collect all data transmitted on both the server and the client. We call this data the *Non-Local* dataset.

We then repeat this process using the same container setup, but across the Docker virtual network on a local machine. We repeat this process several times, generating several *Local* datasets under a variety of emulated network conditions, discussed in Section 4.2. Our *Local* datasets vary slightly in size, but are all roughly 800 megabytes with 245000 packets.

Methodology. NetEm allows us to introduce packet delays according to a variety of distributions, namely uniform, normal, Pareto and Paretonormal[1]. Furthermore, NetEm adds delays according to modifiable distribution tables, and so it is trivial for us to add a Weibull distribution, which along with Pareto distributions have been shown to closely model packet IATs [**? ?**]. In total, we test the efficacy of four distributions to model inter-arrival times — normal, Pareto, Paretonormal and Weibull.

We generate several *Local* datasets by delaying traffic according these distributions, performing an exhaustive grid search over their means and standard deviations. Initial experiments revealed that introducing delays with a mean in the range of 40 ms to 70 ms produced the best results. Setting the jitter of the distribution too high resulted in the repeated arrival of packets out of order, therefore we further limit the grid search to jitter values in 5ms intervals up to half of the value of the mean. In total, we generate 88 *Local* datasets.

Our goal is to discover the *Local* dataset whose packet timings most closely resemble those of our *Non-Local* dataset. To do this, we extract the IATs and packet sizes from our datasets on a packet-by-packet basis and store these results in arrays. We measure the similarity between two of these arrays by training a Random Forest classifier to distinguish between them. We say that if the Random Forest correctly classifies each packet with a success rate of only 50% then it is no better than randomly guessing and, as such, the inter-arrival times of these two arrays are indistinguishable from one another for the Random Forest.

To perform this measurement, we concatenate one *Local* dataset array with our *Non-Local* dataset array, label the entries and then shuffle the rows. We proportion this data into a training set and a testing set using an 80-20 split. We then feed this training data into a Random Forest with 1000 trees and fixed seed, and then record the accuracy of this Random Forest on the test set. We repeat this process for every single *Local* dataset.

Results. Table 2 summarizes the values of the mean and jitter for a given distribution that produced the worst results from the random forest classifier.

Distribution	Mean	Jitter	RF Accuracy
No Delays (Baseline)	0	0ms	0.8176
Constant Delay	40ms	0ms	0.6730
Normal	60ms	5ms	0.6028
Pareto	60ms	10ms	0.5979
Paretonormal	50ms	10ms	0.6015
Weibull	60ms	10ms	0.5540

Table 2: Worst Random Forest accuracy rates for a given distribution

[1]This Paretonormal distribution is defined by the random variable $Z = 0.25 * X + 0.75 * Y$, where X is a random variable drawn from a normal distribution and Y is a random variable drawn from a Pareto distribution.

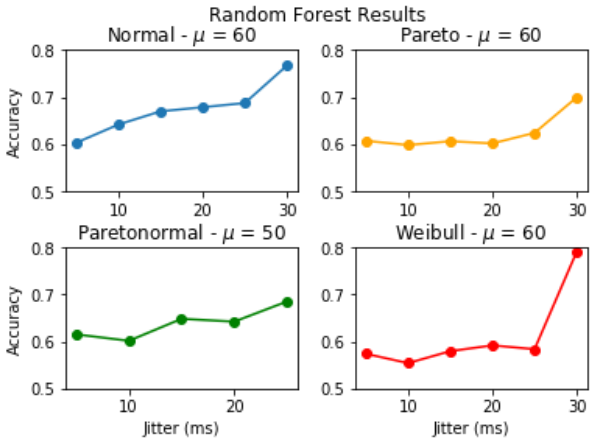

Figure 4: Results of Random Forest Classifier for a given distribution at the best performing delay mean μ. Note that a score of .5 indicates total indistinguishability.

To establish a baseline, we compare the traffic generated from our Docker scenario to that of the *Google Compute* data with no added delays. In this case, the Random Forest was able to distinguish between the two datasets, achieving an accuracy of over 90%. The classification accuracy is worsened considerably by introducing network delays, with the best results being achieved using a Weibull distribution with a mean of 60 ms and a jitter of 10 ms, leading to an accuracy of just 55%. Results for Pareto and Weibull distributions seem to yield consistent results for differing jitter values. Although not completely indistinguishable, this proves that using NetEm we can emulate WAN properties very closely.

5.3 Advantages of Dynamic Dataset Generation

Having examined whether our Docker framework is capable of emulating real-world IATs, we explore their utility in traffic classification to demonstrate the advantages that our framework provides compared to static, unlabeled datasets.

Machine-learning techniques are a popular tool for traffic classification, with many successful published classifiers. Furthermore, inter-packet arrival times have been shown to be a discriminative feature [**? ?**]. However, these methods considered datasets consisting of completed traffic flows, limiting their use in, say, a stateful packet inspector. On-the-fly classifiers are also successful. Jaber et al. [**?**] showed that a K-means classifier can classify flows in real-time solely based on IATs with precision exceeding 90% for most protocols within 18 packets. Similarly, Bernaille et al. [**?**] demonstrated that a K-means classifier can precisely classify traffic within five packets using only packet size as a feature.

However, Jaber et al. [**?**] only evaluated their traffic classifier with training and testing data drawn from the same dataset containing traces of a single network; there is no measure of how this model may generalize to other networks with differing conditions. Furthermore, they were limited to using unsupervised machine learning algorithms to classify their traffic as their datasets had no ground truth.

We attempt to replicate these results within our Docker framework with some adjustments. As we can generate a fully accurate ground truth, we attempt to segregate application flows based on their packet IATs using supervised learning techniques. Moreover, we then measure this model's ability to generalize by expanding our dataset to include traffic from networks with differing bandwidth and latency.

Data & Preprocessing. Our goal is to measure a classifier's ability to generalize across datasets. Therefore we construct two datasets using our Docker framework, both containing the same number of network traces from the same containers.

For our first dataset, we generate `.pcap`-files, each containing traffic from one of 16 different classes: HTTP (Client & Server), HTTPS (Client & Server), RTMP (Client, Server & Viewer), SSH (Client & Server), FTP (Client & Server), IRC (Client & Server), SMTP, SQLi and DoS traffic. To prevent class imbalance, we generate 200 `.pcap`-files for each of the 16 classes, resulting in 3200 total files. To more accurately emulate potential network conditions, we use our NetEm scripts to apply a unique delay to every container involved in a scenario. These delays follow a Pareto distribution with random mean between 0 and 100 milliseconds and random jitter between 0 and 10 milliseconds. We then preprocess this data by removing all but the first 12 packets of each `.pcap`-file. We extract the 11 inter-arrival times separating the 12 packets, which act as our feature vectors. We collect these feature vectors for each class along with a class label, and store collected feature vectors from all 3200 `.pcap`-files in a 12 x 3200 array. We call this our *Primary* dataset.

We then repeat this process to generate a second dataset, changing the properties of our emulated network. Again, we delay all traffic using a Pareto distribution, however, this time we select a random mean in the range of 100 to 500 milliseconds and random jitter between 0 and 50 milliseconds. The subsequent preprocessing of our data remains unchanged. We call this our *Secondary* dataset.

Methodology. First, we attempt to reproduce the results presented by Jaber et al. [**?**] by training a Random Forest with 100 trees to classify application flows based on packet IATs. We do this by proportioning our Primary dataset into training and testing sets using an 80-20 split. We then train and test our Random Forest repeatedly, first considering the classification accuracy based on the IATs of only the first two packets, then the first three packets and so on, up to 12 packets. We record the resulting confusion matrix for each round and calculate the precision and recall rates of our classifier.

Having trained the classifier, we measure its ability to generalize by repeating the above experiment, but replacing the test set with the Secondary dataset.

Results. After each run of our Random Forest on our Primary dataset, we gather the True Positive (T_P), False Positive (F_P) and False Negative (F_N) rate for each class. We then calculate their precision, defined as $\frac{T_P}{T_P+F_P}$, and recall, defined as $\frac{T_P}{T_P+F_N}$, values. in Fig. 7, we see that our average precision and recall across the classes exceeds 0.9 after 10 IATs. Furthermore, after 12 packets our DoS and SQLi data is classified with precision and recall rates of 1.0 and 1.0 and 0.9462 and 0.9322 respectively.

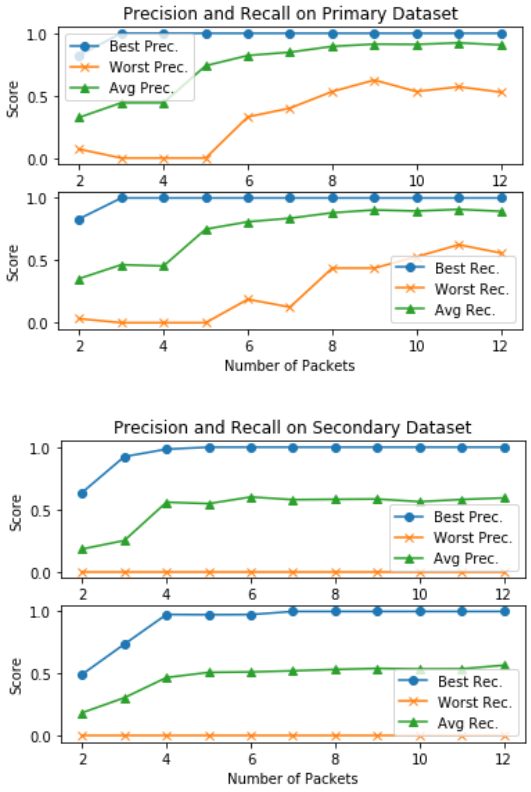

Figure 5: Results of Random Forest Classification on Primary dataset (Above) and Secondary dataset (Below)

These results does not hold when we test the classifyer on our Secondary dataset. As seen in Fig. 7, we see a substantial decrease in our average precision and recall rates, achieving a maximum of 0.5923 and 0.5676 respectively. Moreover, after four packets, increasing the number of IATs in our dataset provides little additional benefit. Although some services generalized well, such as IRC-client and IRC-server, others failed to be classified, with every single SMTP feature being classified as HTTP-client. We also see a substantial drop-off in the classification of malicious traffic, with the precision rates of DoS and SQLi data not exceeding 0.6.

These diverging results demonstrate the necessity of dynamic dataset generation for evaluation purposes. Researchers evaluating their methods only on a dataset with fixed properties such as the Primary dataset might receive overoptimistic results. The capability of generating two or more datasets with the same traffic classes, but otherwise differing properties, provides a more realistic evaluation.

6 CONCLUSIONS

In this paper, we outlined four requirements a modern dataset has to fulfil to strengthen the training of intrusion detection systems. We then proposed a Docker framework capable of generating network intrusion datasets that satisfy these conditions. The major design advantage of this framework are the isolation of traffic scenarios into separate container arrangements, which allows the extension of new scenarios and detailed implementation of subscenarios as well as the capture of ground truth of the computational origins of individual traffic events. Furthermore, containerization enables the generation of traffic data at scale due to containers being lightweight and easily clonable.

We verified the realism of the generated traffic and the corresponding ground truth information with two experiments, and demonstrated the usefulness of the framework in another experiment. Presently, our framework consists of 29 scenarios capable of producing benign and malicious network traffic. Several of these scenarios, such as the *BitTorrent* or the *Stepping-Stone* scenario, provide novel traffic data of protocols or behaviours that has not been widely available to researchers previously.

6.1 Difficulties and limitations

Our framework is building network traffic datasets from a small-scale level up by coalescing traffic from different fine-grained scenarios together. While this provides great insight into small-scale traffic structures, our framework will not replicate realistic network-wide temporal structures, such as port usage distributions or long-term temporal activity. These quantities would have to be statistically estimated from other real-world traffic beforehand to allow our framework to emulate such behavior reliably. Other datasets such as UGR-16 use this approach to fuse real-world and synthetic traffic and are currently better suited to build models of large-scale traffic structures.

Working with Docker containers can sometimes complicate the implementation of individual scenarios compared to working with VMs. Although several applications are officially maintained Docker containers that are free from major errors, many do not. For instance, in the *BitTorrent* scenario, most common command line tools, such as `mktorrent`, `ctorrent` and `buildtorrent`, failed to actually produce functioning torrent files from within a container due to Docker's union filesystem. Furthermore, due to the unique way in which we are using these software packages, unusual configuration settings are sometimes needed.

Lastly, capturing `.pcap`-files from each container can quickly exceed available disc space when generating traffic at scale. Depending on specific research requirements, it is advizable to add filtering or feature extraction commands to the scenario execution scripts to enable traffic preprocessing in real-time.

6.2 Future work

Our traffic generation framework is designed to be expandable and there are many avenues for future work. The continual development of scenarios and subscenarios would improve the potential realism of datasets generated using the framework. The addition of more malicious scenarios would enable a more detailed model evaluation and improve detection rate estimation. Another future improvement for framework is to add scripts that emulate the usage activity of individual scenarios by a user or a network.

Although ground truth for particular traffic traces is provided by capturing `.pcap`-files for each container individually, we have not implemented a labelling mechanism yet for the dataset coalescence process. Though not technically difficult, some thought will

have to be put how such labels would look like to satisfy different research demands. Furthermore, the Docker platform provides the functionality to collect system logs via the `syslog logging driver`. We plan on implementing their collection in the future, where they could act either as traffic labels providing more ground truth details, or act as a separate data source that complements the collected traffic.

We wish to publish this framework to a wider audience, allowing for further modification. This will be done using a GitHub repository, which contains both the implemented capture scenarios as well as the corresponding container images.

7 ACKNOWLEDGMENTS

We are grateful to British Telecommunications PLC who are supporting the PhD research of the first author in the UK EPSRC CASE scheme, giving invaluable guidance on the needs and possibilities of intelligent security tools and their evaluation. Part of this paper draws on the 2019 MSc dissertation of the second author. Nikola Pavlov helped with some of the earlier implementation. The third author was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1 and the Office of Naval Research ONR NICOP award N62909-17-1-2065.