Contribution Title*

First Author[1][0000−1111−2222−3333], Second Author[2,3][1111−2222−3333−4444], and Third Author[3][2222−−3333−4444−5555]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. We introduce DetGen, tool that generates traffic to improve the ability to probe and understand model behaviour in data-driven network intrusion detection, and help explain the corresponding decisions made by a model. DetGen operates under a new design paradigm based on containerisation and reproducibility in order to closely controlling different factors that influence generated network traffic and providing cross-linkage information between captured traffic and these factors. In this work, we demonstrate how DetGen operates.

- We examine how well DetGen is able to control different types of traffic characteristics, and compare the corresponding determinism to common VM-based traffic generation setups.
- We also examine the performance of DetGen in the other direction, namely the ability to generate traffic with realistic levels heterogeneity, and compare the results against those observed in existing artificially generation NID-datasets.
- We present an exemplary dataset that is suitable for a broad probing of models trained on the CICIDS-17 dataset, as it mirrors its range of protocols and attacks.
- We demonstrate the extensive probing of an LSTM-based anomaly detection model with this dataset, and demonstrate how to lower false-positives effectively by understanding where the model fails to process particular traffic structures correctly.

1 Introduction

In this work, we introduce DetGen, a new traffic generation tool that focuses on the ability to probe data-driven traffic models and associate model behaviour with ground-truth descriptive traffic information (or with specific traffic microstructures).

Data-driven traffic analysis and attack detection is a centrepiece of network intrusion detection research, and the idea of training systems on large amounts of network traffic to develop a generalised notion of bad and benign behaviour

* Supported by organization x.

appears like the solution to cyber-threats and has received *tremendous* attention in the academic literature. However, existing datasets fall short on providing any ground-truth information about specific traffic characteristics. Furthermore, the generation process for synthetic traffic usually does not provide sufficient structural nuances to explore the behaviour of models.insert citation

Machine-learning breakthroughs in other fields have often been reliant on a precise understanding of data structure and corresponding descriptive labelling to develop more suitable models. Initial models in *automatic speech recognition (ASR)* for example were reliant on highly sanitised and structured speech snippets before the understanding of these structures lead to the success of more layered models. Lately, datasets that contain labelled specialised speech characteristics enable researchers to better understand ASR weak points such as emotional speech (RAVDESS), accents (Speech Accent Archive), or background noise (Urban Sound Dataset).

In a similar fashion, several approaches to enhance the way information is collected and presented have been successful in improving understanding between data and cyber-detection systems. Virtual machine introspection monitors and analyses the runtime state of a system-level VM, and the inclusion of threat reports to create behavioural feature labels enriches the way executables are described [14]. However, such efforts have not been made in network intrusion detection yet, with the current quasi-benchmark datasets paying more attention to the inclusion of a wide variety of attacks rather than the close control and detailed documentation of the generated traffic, which has so far lead researchers to predominantly apply a number of ML-models to traffic datasets in the hope of edging out competitors.

This work provides the following contributions:

1. We describe how the container-based design paradigm of DetGen provides accurate and quasi-deterministic control over traffic characteristics as well as corresponding ground truth information, and compare the design advantages to traditional generation set-ups.
2. We examine how well DetGen is able to control different types of traffic characteristics, and compare the corresponding determinism to common VM-based traffic generation setups.
3. We present an exemplary dataset that is suitable for a broad probing of models trained on the CICIDS-17 dataset, as it mirrors its range of protocols and attacks.
4. We demonstrate the extensive probing of an LSTM-based anomaly detection model with this dataset, and demonstrate how to lower false-positives effectively by understanding where the model fails to process particular traffic structures correctly.

This framework is openly accessible for researchers and allows for straightforward customization.

1.1 Outline

2 Motivation and Methodology

Scientific machine learning model development requires both **model evaluation**, in which the overall predictive quality of a model is assessed to identify the best model, as well as **model validation**, in which the behaviour and limitations of a model is assessed through targeted **model probing**. Model validation is essential to understand how particular data structures are processed, and enables researchers to develop their models accordingly. Data generation tools for rapid model probing in other domains such as the *What-If tool* [19] underline the importance of model validation.

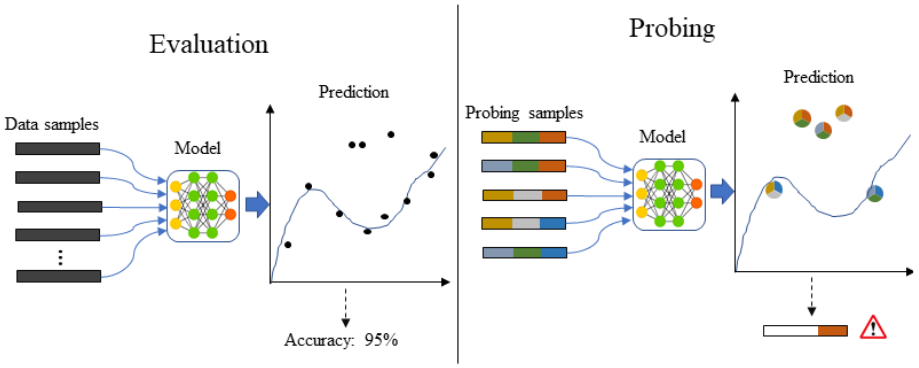


Fig. 1: Comparison between numerical model evaluation and model probing with specifically controlled data characteristics, indicated as colours.

2.1 DetGen-scope

Assume the following problem: You are designing a packet-level traffic classifier for a specific setting, which is generating a significant amount of misclassification. These turn out to be caused by a particular characteristic such as an unsuccessful login or frequent connection restarts. However, existing real-world or synthetic datasets do not contain the necessary information to associate the misclassified traffic events with these characteristics, which prevents you from identifying the misclassification cause and the corresponding model design flaw purely by numeric evaluation.

DetGen is designed to provide traffic with the necessary ground-truth information to allow effective association of model processing behaviour with corresponding traffic characteristics. Traffic is generated with precise control and

monitoring over the conducted communication activity as well as various traffic-shaping factors. The scope lies on microscopic traffic-structures that become apparent on a packet-individual-flow-level.

DetGen separates program executions and traffic capture into distinct containerised environments in order to exclude any background traffic events and therefore provide precise ground-truth about the computational origin of the traffic, something that is lacking in all network traffic datasets that we are aware of. By using containerisation, we are furthermore able to control and shield the traffic generation better than in conventional VM-setups, as we demonstrate in Section ..., and consequently provide better reproducible data.

In order to examine additional effects on traffic micro-structures, DetGen simulates influence factors such as network congestion, communication failures, data transfer size, content caching, or application implementation.

Below, we provide a use-case to demonstrate how ground-truth information and traffic micro-structure-control enables effective model probing:

2.2 Improved traffic separation for a classifier with congestion level information

In this example we improve the design of a traffic classification model through the analysis of data separation in dependence of different traffic features. For this, we use a recent traffic classification model by Hwang et al. [7] as an example, which aims at distinguishing various types of malicious activity from benign traffic. The model achieved some of the highest detection rates of packet-based classifiers in a recent survey [17]. The model classifies connections on a packet-level using a *Long-short-term memory* (LSTM) network [4], and is claimed to achieve detection and false-positive (FP) rates of **99.7%** and **0.03%** respectively.

We train a model on a set of different HTTP-activities generated by DetGen in order to detect SQL-injections. Rather than providing an accurate and realistic detection setting, this example shows how traffic information can be linked to model failures and slumping performance. We use real-world HTTP-traffic from the *CAIDA anonymized traffic traces* [18] as background traffic (85% of connections) and add SQL-injection attack traffic (7.5%) as well as different HTTP-activities for probing (7.5%). In total, we use 50,000 connections for training the model, or slightly less than 2 million packets.

The initially trained model performs relatively well, with an *Area under curve* (AUC)-score[5] of **0.981**, or a detection and false positive rate[6] of **0.96%** and **2.7%**. However, these rates are still far from enabling operational deployment. Now suppose we want to improve these rates to both detect more SQL-injections and retain a lower false-positive rate.

We initially explore which type of connections are misclassified most often. For this, we perform a correlation analysis between the numeric or categoric

[4] a deep learning design for sequential data
[5] a measure describing the overall class separation of the model
[6] tuned for the geometric mean

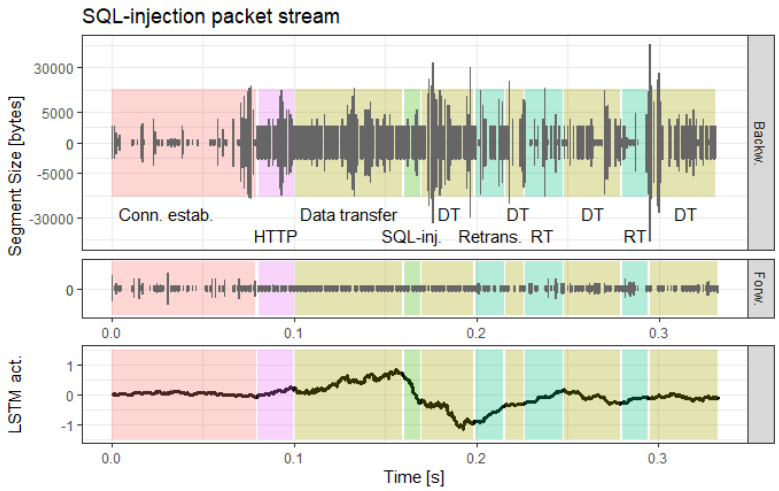


Fig. 2: LSTM-output activation in dependence of connection phases. Depicted are packet segment streams and their respective sizes in the forward and backward direction, with different phases in the connection coloured and labelled. Below is the LSTM-ouput activation while processing the packet streams.

labels available for the probing data, and the binary response whether the corresponding connection was misclassified. Unsurprisingly, the highest correlation to misclassification was measured for the conducted activity, with a particular attack scenario (19% correlation) and connections with multiple GET-requests (11% correlation) being confused most often. This was followed by the amount of simulated latency (12% correlation), which we are now examining closer.

Fig. 3 depicts classification scores of connections in the probing data in dependence of the emulated network latency. The left panel depicts the scores for the initially trained model, which shows that while classification scores are well separated for lower congestion, increased latency in a connection leads to a narrowing of the classification scores, especially for SQL-injection traffic. Since there are no classification scores that reach far in the opposing area, we conclude that congestion simply makes the model lose predictive certainty. Increased latency can both increase variation in observed packet interarrival times (IATs), and lead to packet out-of-order arrivals and corresponding retransmission attempts. Both of these factors can decreases the overall sequential coherence for the model, i.e. that the LSTM-model loses context too quickly either due to increased IAT variation or during retransmission sequences.

To examine the exact effect of retransmission sequences on the model output, we generate two similar connections, where one connection is subject to moderate packet loss and reordering while the other is not. We then compare how the LSTM-output activation is affected by retransmission sequences. Fig. 2 depicts the evolution the LSTM-output layer activation in dependence of difference con-

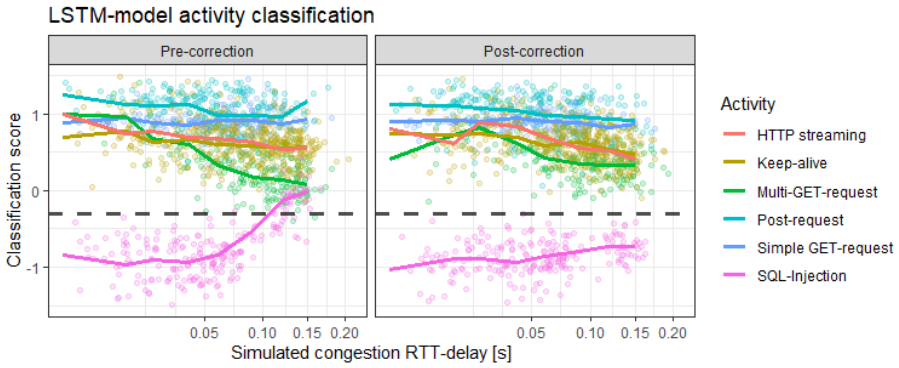


Fig. 3: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold.

nection phases. Initially the model begins to view the connection as benign when processing regular traffic, until the SQL-injection is performed. The model then quickly adjusts and provides a malicious classification after processing the injection phase and the subsequent data transfer. The negative output activation is however quickly depleted once the model processes a retransmission phase, and is afterwards not able to relate the still ongoing data transfer to the injection phase. When comparing this to the connection without retransmissions, we do not encounter this depletion effect, instead the negative activation persists after the injection phase.

We try to correct the existing model with a simple fix by excluding retransmission sequences from the model input data, both during training and classification. This leads to significantly better classification results during network latency, as visible in the right panel of Fig. 3. SQL-injection scores are now farless affected by congestion while scores for benign traffic are also less affected, albeit to a smaller degree. The overall AUC-score for the model improves to **0.997** while tuned detection rates and false positives improved to **99.1%** and **0.045%**.

3 Probing dataset

Do we need a name for this dataset?

We compiled and released a dataset suitable to quickly probe ML-model behaviour on a range of traffic characteristics. This dataset is designed to contain similar traffic as the CICIDS-17 and 18 datasets to allow probing of models trained on these de-facto benchmark datasets.

3.1 Mirroring CICDS-17 dataset

Our dataset is compiled to enable quick probing of models trained on the CICIDS-17 or 18 datasets. For this, we mirrored several high-level properties to provide the same traffic structures the models learned. In particular, we mirrored the following properties:

- **Application layer protocols**: We used the same range of major protocols that occur in the CICIDS-datasets, namely *HTTP/SSL, SMTP, FTP, SSH, SQL, SMB, and NTP*. We excluded some protocols such as DNS, LDAP, or Kerberos since these do not occur in sufficient amount and complexity in the CICIDS-datasets. Table in Appendix displays the corresponding frequency of protocols.
  HTTP/HTTPS NTP 2 packets SMB print LDAP Kerberos 3.5 packets RPC SSDP
- **Implementations**: For each of the protocols, we used similar application implementations, such as Apache for HTTP-activities ...
- **Conducted attack types**: We aimed to cover as many attack types inlcuded in the CICIDS datasets as possible. These include *SQL-injections, SSH-brute-force, XSS, Botnet, Heartbleed, GoldenEye, and SlowLoris*. We were not able to cover all attacks though as DetGen either did not provide the necessary network topology to conduct the attack, such as for port-scanning, or the attack types are not implemented in the catalogue of scenarios yet.
- **Activity range**: Especially for HTTP-traffic, we included the same types of activities ...

In addition to that, we used the same tool, the *CICFlowMeter* to generate the 83 flow-features.

In Fig. ... we compare the validation error of a recent neural network model on the probing dataset, when trained exclusively on the CICIDS data, and when also trained on the probing data. Even though the validation error is slightly higher when only trained on the CICIDS data, the difference is almost negligible compared to the error resulting from a model trained on a completely different dataset (UGR-16). This does not fully proof that every model is able to transfer observed structures between the two datasets, but it gives an indicator that they mirror characteristics.

3.2 Dataset statistics

To be filled

4 DetGen and Traffic-Microstructures

4.1 Influence factors on traffic-microstructures

In order to enable sufficient and reproducible control over the generated traffic and provide the corresponding descriptive ground truth information, we first

must understand what factors shape the traffic generation process. Computer communication involves a myriad of different computational aspects, and no research so far has been conducted to quantify how much influence each of them has on traffic structures. We highlight and quantify the influence of the most important influence factors on the traffic micro-structures observed on individual devices. These will act as a justification for the design choices we outline in Section ....

1. Application layer protocols Without doubt the biggest impact on the captured traffic micro-structures is the choice or combination of the application layer protocols. Protocols such as HTTP/TLS perform vastly different tasks than protocols such as Peer-2-Peer or SMB, and thus perform different handshakes, experience different waiting times, transfer data in different intervals, or trigger different additional connections.

2. Performed task and application The conducted computational task ultimately drives the communication between computers, and thus hugely influences characteristics such as the direction of data transfer, the duration and packet rate, packet sizes as well as the number of connections and performed protocol handshakes to conclude the task. Furthermore, the application used for the task has a significant influence on the generated traffic, as shown for different browser choices by Yen et al. [20] or for general application behaviour fingerprinting [16].

3. Transferred data The amount of transferred data influences the overall packet numbers. Furthermore, the content of the data can potentially impact packet rates and sizes, such as shown by Biernacki [2] for streaming services.

4. Caching/Repetition effects Tools like cookies, website caching, DNS caching, known hosts in SSH, etc. remove one or more information retrieval requests from the communication, which can lead to altered packet sequences, less connections being established. For caching, this can result in less than 10% of packets being transferred, as shown by Fricker et al. [3].

Time	Source-IP	Destination-IP	**Dest. Port**
13:45:56.8	192.168.10.9	192.168.10.50	**21**
13:45:56.9	192.168.10.9	192.168.10.50	**10602**
13:45:57.5	192.168.10.9	69.168.97.166	**443**
13:45:59.1	192.168.10.9	192.168.10.3	**53**
13:46:00.1	192.168.10.9	205.174.165.73	**8080**

Table 1: Exemplary activity interval for host 192.168.10.9 in the CICIDS-17 dataset, containing FTP-, HTTPS- and DNS-, as well as additional unknown activity.

5. Captured traffic from background activity In traditional setups, all traffic generated on a host is recorded in the same capture, which makes it hard if not impossible to disentangle traffic from different activities and match them to their origin. Capturing background traffic typically leads to additional flows within the given time interval. 74% of SSH-connections and more than 95% of FTP- and HTTPS-connections in the CICIDS-17 dataset lie within a 5-second interval of connections from other background activity on the same network interface, as depicted in Table 1.

6. Application layer implementations Different implementations for TLS, HTTP, etc. can yield different computational performance and can perform handshakes in slightly different ways. Furthermore, things like multiplexing channel prioritisation can have tremendous impact on the IAT times and the overall duration of the transfer, as shown in a study by Marx et al. [11] for the QUIC/HTTP3 protocol.

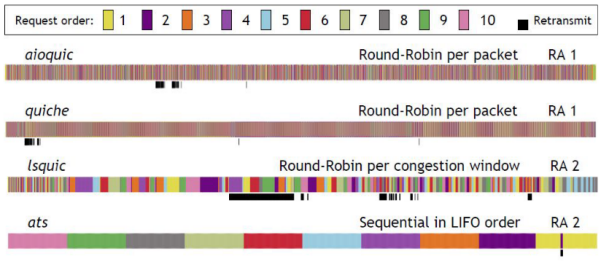


Fig. 4: Comparison of QUIC connection request multiplexing for selected implementations, taken from [11].

7. Host level load In a similar manner, other applications exhibiting significant computational load (CPU, memory, I/O) on the host machine can affect the processing speed of incoming and outgoing traffic, which can again alter IATs and the overall duration of a connection. An example of this is visible in Fig. 5, where the host sends significantly less ack-packets when under heavy computational load.

8. LAN and WAN congestion Low available bandwith, long RTTs, or packet loss can have a significant effect on TCP congestion control mechanisms, which in turn influence frame-sizes, IATs, window sizes, and the overall temporal characteristic of the sequence. do we need to verify this? Seems very clear

We designed DetGen to control and monitor these factors in order to let researchers explore the impact of different aspects on their traffic models. We omitted some factors that can influence traffic structures, since these act either

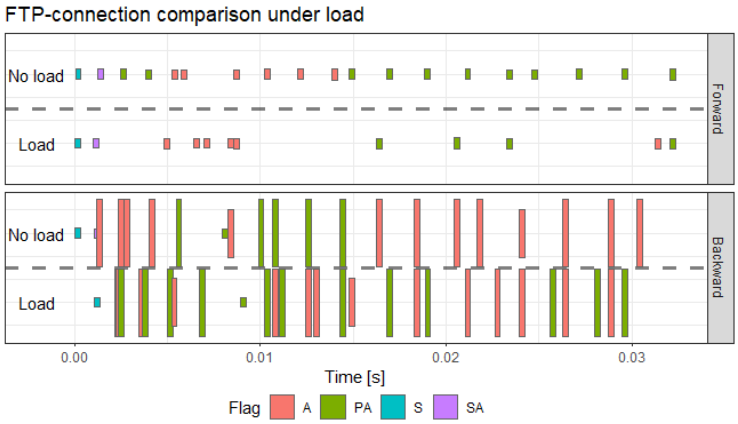


Fig. 5: Packet-sequence structure similarity comparison for FTP-activity under different load and otherwise constant settings. Colours indicate packet flags while the height of the packets indicates their size. Note that under load, the host sends significantly less packets.

on a larger scale rather than micro-structures or correspond to exotic settings that are outside of our traffic generation scope. Among them are the following:

1. User and scheduled activities The choice and usage frequency of an application and task by a user governs the larger-scale temporal characteristic of a traffic capture. Since we are focusing on the traffic micro-structures here, we currently omit this impact factor from our analysis.

2. Networking stack load TCP or IP queue filling of the kernel networking stack can increase packet waiting times and therefore IATs of the traffic trace, as shown by [12]. In practice, this effect seems to be constrained to large WAN-servers and routers, and we did not find any effect on packet-sequences for various amounts of load generated with iPerf for a regular UNIX host.

3. Network configurations Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes, are standardised for most networks where to find a proof for that?.

4.2 Generative determinism of DetGen

4.3 Traffic control and generation determinism

We now assess the claim of control over the outlined traffic influence factors, and how similar traffic generated with the same settings looks like. We also demonstrate that this level of control is not achievable on regular VM-based NIDS-traffic-generation setup.

To do so, we generate traffic from settings within which all controllable influence factors are held constant, both with DetGen framework and with a regular VM-based setup. Traffic samples from each setting should then be as similar as possible to provide sufficient experimental determinism. To measure how similar two traffic samples are, we devise a set of similarity metrics that measure dissimilarity of overall connection characteristics, connection sequence characteristics, and packet sequence characteristics:

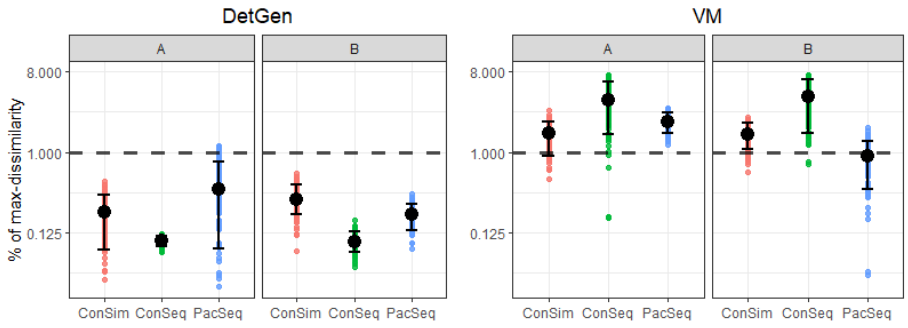


Fig. 6: Comparison of HTTP-group dissimilarity scores for the DetGen-framework and a regular VM-setup, on a logarithmic scale. Samples from the VM-setting are consistently more dissimilar, in particular for flow-based metrics, where the average dissimilarity is more than 30 times higher than for the DetGen setting.

- **Overall connection similarity** We collect use the 82 flow summary statistics (IAT and packet size, TCP window sizes, flag occurrences, burst and idle periods) provided by CICFlowMeter insert citation, and measure the cosine similarity between connections, which is also used in general traffic classification [**?**].
- **Connection sequence similarity** To quantify the similarity of a sequence of connections in a retrieval window, we use the following features to describe the window, such as used by Yen et al. [20] for application classification: The number of connections, average and max/min flow duration and size, number of distinct IP and ports addresses contacted. We then again measure the cosine similarity based on these features between different windows.
- **Packet sequence similarity** To quantify the similarity of packet sequences in traffic captures, we assign packets a discrete state according to their flags, direction, sizes, and interarrival times (insert citation). We then calculate the Markovian probability of each packet state conditional on the previous packet. We do this for sequences of 15 packets at the start, the middle, and the end of a connection, and use the average sequence likelihood of each

group as a similarity measure. If connections are completely similar, the conditional probabilities and thus the likelihoods should converge to one.

We normalise all dissimilarity scores by dividing them by the maximum dissimilarity score measured for each traffic type in our experiment in Section **??**, so that the reader can relate the measured scores to the traffic type.

As a comparison, we use a regular VM-based setup, were applications are hosted directly on two VMs that communicate over a virtual network bridge that is subject to the same NetEm effects as DetGen. To compare the amount of traffic control and the corresponding generative determinism of DetGen and the VM-setup, we generate three different types of traffic (HTTP, file-syncing, and botnet) from four different settings, within which all generative parameters are kept constant. For each setting and traffic type, we generate 100 traffic samples and apply the described dissimilarity measures to 100 randomly drawn pairs sample pairs. Fig. 6 depicts the calculated dissimilarity scores for DetGen and the VM-setup.

The scores yield less than 1% of the dissimilarity observed on average for each protocol. Scores are especially low when compared to traffic groups collected in the VM setting, which is also visible in Fig. **??** for the HTTP-traffic. Dissimilarity scores for the VM-setting are most notably higher for the flow-metric, caused by additional background flows frequently captured. While sequential dissimilarity is roughly the same for the DetGen- and the VM-settings, overall connection similarity for the VM-setting sees significantly more spread in the dissimilarity scores when computational load is introduced.

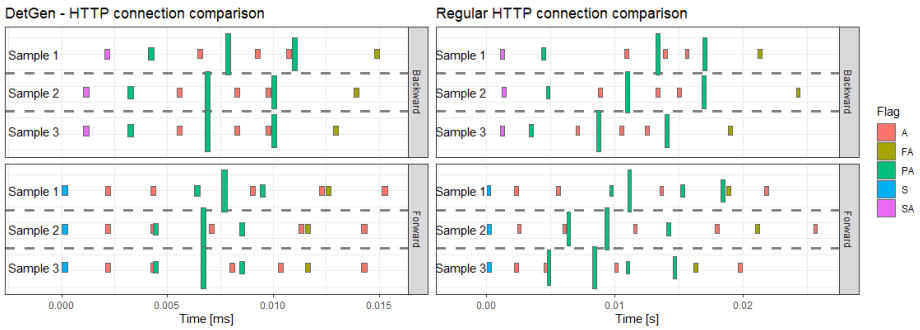


Fig. 7: Packet-sequence structure similarity comparison for HTTP-activity under constant settings generated by the DetGen framework (left) and in a regular setting (right). Colours indicate packet flags while the height of the packets indicates their size. Note that in addition to more differences in the timing, the packet sizes vary more in the regular setting.

5 DetGen Architecture

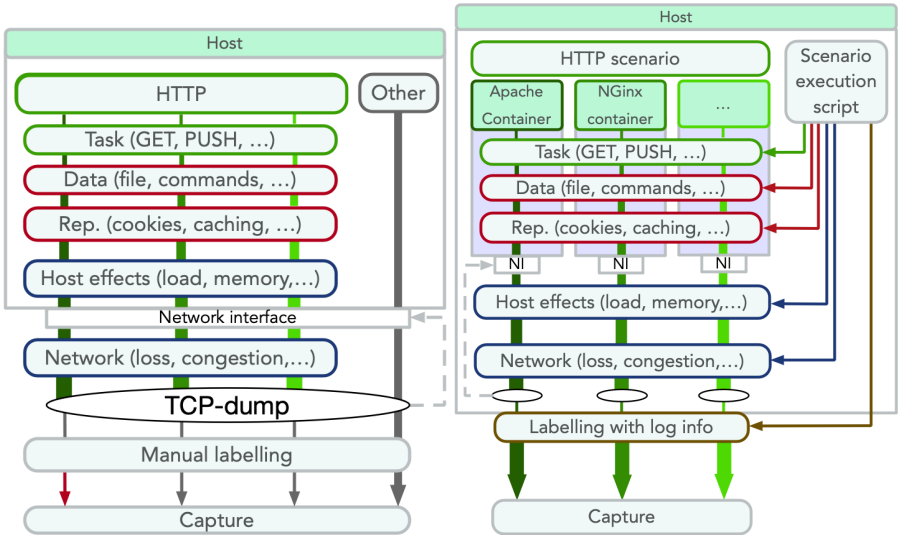
5.1 Design overview

Detgen is a container-based network traffic generation framework that dis. In contrast to the pool of programs running in a VM-setup, DetGen separates program executions and traffic capture into distinct containerised environments in order to shield the generated traffic from external influences and enable the fine-grained control of traffic shaping factors.

Traffic is generated from a set of scripted *scenarios* (give examples here) that strictly control corresponding influence factors and offer the researcher to modify and label the conducted activity from a variety of angles and randomisations.

5.2 Containerization and activity isolation

Containers are standalone packages that contain an application along with all necessary dependencies using OS-level virtualization. In contrast with standard Virtual machines (VMs), containers forego a hypervisor and the shared resources are instead kernel artifacts that can be shared simultaneously across several containers, leading to minimal CPU, memory, and networking overhead [9].



(a) Traditional capture setup (b) DetGen container setup

Fig. 8: Design comparison of traditional NIDS-data-setups and our DetGen framework

Due to the separation of processes, containers provide significantly more isolation of programs from external effects than regular OS-level execution. This

isolation enables us to monitor processes better and create more accurate links between traffic events and individual activities than on a virtual machine were multiple processes run in parallel, which can all generate traffic. The corresponding one-to-one correlation between processes and network traces allows us to produce labelled datasets with significantly more granular ground truth information.

Insert some experimental result here.

Containers are specified in an image-layer, which is unaffected during the container execution. This allows containers to be run repeatedly whilst always starting from an identical state. In combination with the container isolation, this allows us to perform network experiments that can be easily reproduced by anyone on any plattform insert citation.

5.3 Activity generation

Scenario We define a *scenario* as a series of Docker containers conducting a specific interaction, whereby all resulting network traffic is captured from each container's perspective. This constructs network datasets with total interaction capture, as described by Shiravi et al. [13]. Each scenario produces traffic from a specific setting with two (client/server) or more containers. Examples may include an FTP interaction, a music streaming application, an online login form paired with an SQL database, or a C&C server communicating with an open backdoor. A full list of currently implemented scenarios can be found in Section 5.6. Each scenario is designed to be easily started via a single script and can be repeated indefinitely without further instructions, therefore allowing the generation of large amounts of data. Our framework is modular, so that individual scenarios are configured, stored, and launched independently. Adding or reconfiguring a scenario has no effect on the remaining framework.

When composing different settings, we most emphasised the inclusion of different **application layer protocols** such as HTTP or SSH, followed by the inclusion of different corresponding **applications** such as NGINX or Apache that steer the communication. We are currently aiming to also include options to use different **application layer implementations** such as TLS1.3 vs TLS1.2.

Task In order to provide a finer grain of control over the traffic to be generated, we create a catalogue of different tasks that allow the user to specify the manner in which a scenario should develop. The aim of having multiple tasks for each scenario is to explore the full breadth of a protocol or application's possible traffic behaviour. For instance, the SSH protocol can be used to access the servers console, to retrieve or send files, or for port forwarding, all of which may or may not be successful. It is therefore appropriate to script a number of tasks that cover this range of tasks.

To implement a catalogue of tasks, we first examine the functionality of the underlying protocol and scenario setting before proceeding to adding tasks to the catalogue. To explore the breadth of the corresponding traffic structures efficiently, we prioritise to add tasks that cover aspects such as direction of file

transfers (e.g. GET vs POST for HTTP), the amount of data transferred (e.g. HEAD/DELETE vs GET/PUT), or the duration of the interaction (e.g. persistent vs non-persistent tasks) as much as possible. For each task, we furthermore add different failure options for the interaction to not be successful (e.g. wrong password or file directory).

Since we always launch containers from the same state, we prevent traffic impact from **repetition effects** such as caching or known hosts. If an application provides caching possibilities, we implement this as an option to be specified before the traffic generation process.

Input randomization Scripting activities that are otherwise conducted by human operators often leads to a loss of random variation that is normally inherent to the activity. As mentioned in Section **??**, the majority of successful FTP transfers in the CIC-IDS 2017 data consist of a client downloading a single text file. In reality, file sizes, log-in credentials, and many other variables included in an activity are more or less drawn randomly, which naturally influences traffic quantities such as packet sizes or numbers.

We identify variable input parameters within scenarios and corresponding tasks and systematically draw them randomly from suitable distributions. Passwords and usernames, for instance, are generated as a random sequence of letters with a length drawn from a truncated Cauchy distribution, before they are passed to the corresponding container. Files to be transmitted are selected at random from a larger set of files, covering different sizes and file names.

5.4 Simulation of external influence

Network effects Docker communication takes place over virtual bridge networks,

Communication between containers takes place over a virtual Mininet bridge network, which provides far higher and more reliable throughput than in real-world networks. Gates and Warshavsky [4] measured a bandwidth of over 90 Gbits/s without any lost packets using iPerf.8 This allows us to guarantee reliable and reproducible communication and thus remove external network effects on the captured traffic.

Virtual bridge networks furthermore enable us to retard and control the network reliability and congestion to a realistic level by using emulation tools. NetEm is an enhancement of the Linux traffic control facilities for emulating properties of wide area networks such as high latency, low bandwidth or packet corruption by adding delay, packet loss, duplication etc. to packets outgoing from a selected network interface [6].

We apply NetEm via a wrapping script to to the network interface of a given container, providing us with the flexibility to set each container's network settings uniquely. In particular, packet delays are drawn from a Paretonormal-distribution while packet loss and corruption is drawn from a binomial distribution, which has been found to emulate real-world settings well [8]. Distribution

parameters such as mean or correlation as well as available bandwidth can either be manually specified or drawn randomly before the traffic generation process.

Host load We simulate excessive computational load on the host with the tool *stress-ng*, a Linux workload generator. Currently, we only stress the CPU of the host, which is controlled by the number of workers spawned. Future work will also include stressing the memory of a system. We have investigated how stress on the network sockets affects the traffic we capture without any visible effect, which is why we omit this variable here.

5.5 Activity execution

Execution script DetGen generates traffic through executing execution script that are specific to the particular scenario. The script creates the virtual network and populates it with the corresponding containers. The container network interfaces of the containers are then subjected to the NetEm chosen settings and the host is assigned the respective load, before the inputs for the chosen task are prepared and mounted to the containers.

The user can then choose how long and how often to execute the scenario. Once the activity is terminated, the script takes down the network and containers, and repeats the process for the next repetition. Randomised settings are drawn anew for each repetition.

Labelling and traffic separation Each container network interface is hooked to a *tcpdump*-container that records the packets that arrive or leave on this interface. Combined with the described process isolation, this setting allows us to exclusively capture traffic that corresponds to the conducted activity and exclude any background events. The captured traffic is then saved and labelled as a pcap-file. The execution script then stores all parameters (conducted task, mean packet delay,...) and descriptive values (input file size, communication failure, ...) for the chosen settings in a file along with the corresponding pcap-filename.

5.6 Existing 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.

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
Wordpress	Client accessing Wordpress site	5
Syncthing	Clients synchronize files via Syncthing	7
mailx	Mailx instance sending emails over SMTP	5
IRC	Clients communicate via IRCd	4
BitTorrent	Download and seed torrents	3
SQL	Apache with MySQL	4
NTP	NTP client	2
Mopidy	Music Streaming	5
RTMP	Video Streaming Server	3
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 2: Currently implemented traffic scenarios along with the number of implemented subscenarios

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.

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 2.

6 Conclusions

In this paper, demonstrated the impact of traffic generation with extensive micro-structure control as well as detailed corresponding documentation on researchers ability to evaluate and understand network intrusion detection models. We implemented and trained two state-of-the art detection models before extensively probing their behaviour and limitations when encountering different traffic types.

By using HTTP-traffic with congestion settings, we were quickly able to identify the inability of an LSTM-based classifier to handle traffic with significant retransmission rates, which enabled us to improve the model accordingly and increase detection performance by more than 2%. Similarly, the examination of projection consistency of a subspace-clustering method using traffic with artificially similar characteristics revealed an overly high sensitivity to flow interarrival times, while cluster-coherence could be increased significantly by identifying half-open connections that were dropped because of network failure as the source of overly dispersed traffic projections.

These results have encouraged us to perform more deep-going probing of data-driven network intrusion detection models. We believe that in combination with strong NID-dataset, extensive model validation and corresponding development with targeted traffic samples might hold the key to reduce false positives of detection models to an acceptable rate, as well as help models replicate detection rates in practical settings.

6.1 Difficulties and limitations

While the control of traffic micro-structures helps to understand models that perform on a packet- or connection-level, it does not replicate realistic network-wide temporal structures, such as port usage distributions or long-term temporal activity. The probing of models operating on aggregated, behavioural, or long-term features is therefore not effective, and variation in these quantities would have to be statistically estimated from other real-world traffic beforehand to allow our framework to emulate such behaviour 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.

Furthermore, while controlling traffic shaping factors artificially helps at identifying the limits and weak points of a model, it can exaggerate some characteristics in unrealistic ways and thus both affect the training phase of a model as well as tilt the actual detection performance of a model in either direction. Additionally, the artificial randomisation of traffic shaping factors can currently not generate the traffic diversity encountered in real-life traffic and thus only aid at exploring model limits extensively. The lack of realistic traffic heterogeneity however is at the moment significantly more pronounced in commonly used network intrusion datasets such as the CICIDS-17 dataset, where the vast majority of successful FTP-transfers consist of a client downloading a single text file that contains the Wikipedia page for 'Encryption'.

6.2 Future work

Import of activity timeline The modelling and generation of computer network activity has been investigated extensively (citations?), and tools to automatically generate realistic network activity streams

we do not wish to address this topic here . Instead, our framework imports existing time-series of host flow activity to generate the corresponding communication. give more info on flow generation tools

We transform existing network flow series into an activity timeline by expand this. We end up with an activity timeline that contains a set of timestamps along with the corresponding scenario and the source and destination host.

References

1. M. Barre, A. Gehani, and V. Yegneswaran. Mining data provenance to detect advanced persistent threats. In *11th International Workshop on Theory and Practice of Provenance (TaPP 2019)*, 2019.
2. A. Biernacki. Analysis and modelling of traffic produced by adaptive http-based video. *Multimedia Tools and Applications*, 76(10):12347–12368, 2017.
3. C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 310–315. IEEE, 2012.
4. M. Gates and A. Warshavsky. Iperf Man Page. `https://linux.die.net/man/1/iperf`. Accessed: 2019-08-11.
5. R. Harang. Bridging the semantic gap: Human factors in anomaly-based intrusion detection systems. In *Network Science and Cybersecurity*, pages 15–37. Springer, 2014.
6. S. Hemminger et al. Network emulation with netem. In *Linux conf au*, pages 18–23, 2005.
7. R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang. An lstm-based deep learning approach for classifying malicious traffic at the packet level. *Applied Sciences*, 9(16):3414, 2019.

8. A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang. An empirical study of netem network emulation functionalities. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2011.
9. K. Kolyshkin. Virtualization in linux. *White paper, OpenVZ*, 3:39, 2006.
10. H. Liu and B. Lang. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences*, 9(20):4396, 2019.
11. R. Marx, J. Herbots, W. Lamotte, and P. Quax. Same standards, different decisions: A study of quic and http/3 implementation diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pages 14–20, 2020.
12. L. Sequeira, J. Fernández-Navajas, L. Casadesus, J. Saldana, I. Quintana, and J. Ruiz-Mas. The influence of the buffer size in packet loss for competing multimedia and bursty traffic. In *2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 134–141. IEEE, 2013.
13. A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.
14. M. R. Smith, N. T. Johnson, J. B. Ingram, A. J. Carbajal, R. Ramyaa, E. Domschot, C. C. Lamb, S. J. Verzi, and W. P. Kegelmeyer. Mind the gap: On bridging the semantic gap between machine learning and information security. *arXiv preprint arXiv:2005.01800*, 2020.
15. R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.
16. T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are? smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 7–12, 2013.
17. H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar. The rise of traffic classification in iot networks: A survey. *Journal of Network and Computer Applications*, 154:102538, 2020.
18. C. Walsworth, E. Aben, K. Claffy, and D. Andersen. The caida ucsd anonymized internet traces 2018,", 2018.
19. J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65, 2019.
20. T.-F. Yen, X. Huang, F. Monrose, and M. K. Reiter. Browser fingerprinting from coarse traffic summaries: Techniques and implications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 157–175. Springer, 2009.