Controlling network traffic microstructures for machine-learning model probing

No Author Given

No Institute Given

Abstract. Network intrusion detection models increasingly rely on learning traffic microstructures that consist of pattern sequences in features such as interarrival time, size, or packet flags. We argue that precise and reproducible control over traffic microstructures is crucial to understand and improve NID-model behaviour. We demonstrate that probing a traffic classifier with appropriately generated microstructures reveals links between misclassifications and traffic characteristics, and correspondingly lets us improve the false positive rate by more than 500%. We examine how specific factors such as network congestion, load, conducted activity, or protocol implementation impact traffic microstructures, and how well their influence can be isolated in a controlled and near-deterministic traffic generation process. We then introduce DetGen, a traffic generation tool that provides precise microstructure control, and demonstrate how to generate traffic suitable to probe pre-trained NID-models.

1 Introduction

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**, as depicted in Fig. 1. 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 such as the *What-If tool* [32] underline the importance of model validation, but are not suitable for providing probing data that resembles the complex structures found in network packet streams.

Machine-learning breakthroughs in many fields have been reliant on a precise understanding of data structure and corresponding descriptive labelling to develop more suitable models. In *automatic speech recognition (ASR)*, tone and emotions can alter the meaning of a sentence significantly. The huge automatically gathered speech datasets however only contain speech snippets and if possible their plain transcripts. While modern speech models are in principle able to learn implicit structures such as emotions without explicit labels, it is impossible to determine the cause for systematic error when they are not. Datasets that contain labelled specialised speech characteristics such as the Ryerson Database of Emotional Speech and Song (RAVDESS) [15] not only allow researchers to

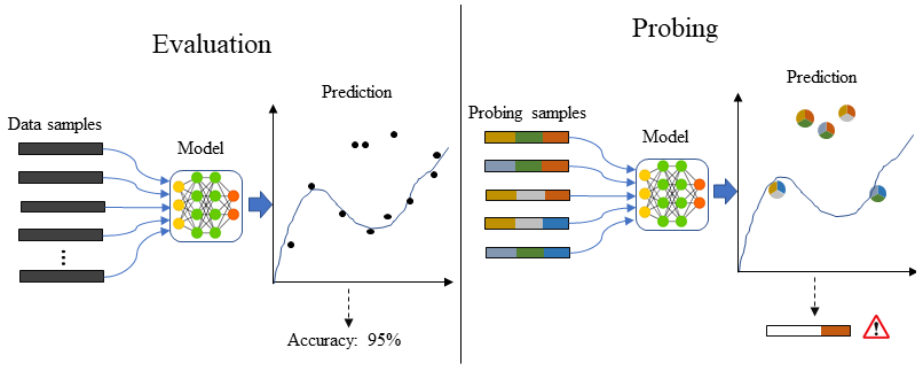

Fig. 1: Model evaluation and model probing with controlled data characteristics.

identify if their model is susceptible to structural misclassification through targeted probing, but also inspire new methods to capture and understand these implicit structures [6], which in turn leads to design improvements of general speech recognition models [10].

Prominent network intrusion detection methods as Kitsune [19] or DeepCorr [21] learn structures in the sizes, flags, or interarrival times of packets for decision-making. These **traffic microstructures** that can be observed when looking sequences of packets or connections reveal information about attack behaviour, but are also influenced by a number of other factors such as network congestion or the transmitted data. However, no effort has been made so far to monitor or control these factors to probe these models for specific microstructures, with the current quasi-benchmark NID-datasets pay more attention to the inclusion of specific attacks and topologies rather than the documentation of the generated traffic. This situation has so far led researchers to often simply evaluate a variety of ML-models on these datasets in the hope of edging out competitors, without understanding model flaws and corresponding data structures through targeted probing.

We demonstrate how to produce traffic effectively to probe a state-of-the-art traffic classifier, and why a certain degree of generative **determinism** is required for this to isolate the influence of traffic microstructures. The model insights and corresponding performance improvements achieved through probing motivate our experimental examination of various influence factors over microstructures and how to control them during traffic generation. Finally, we present *DetGen*, a traffic generation tool that provides near-deterministic control over microstructure-shaping factors such as conducted activity, communication failures or network congestion to generate reproducible traffic samples along with corresponding ground-truth labels.

This work provides the following contributions:

1. We demonstrate why model probing with controllable traffic microstructure is a crucial step to understand and ultimately improve model behaviour

by probing a state-of-the-art LSTM traffic classifier and lowering its false positives five-fold.
2. We discuss requirements for traffic data suitable to probe models pre-trained on a given NIDS-dataset, and demonstrate how to generate probing traffic effectively through DetGen-IDS, a dedicated probing dataset.
3. We examine experimentally how different factors affect traffic microstructures, and how well they can be controlled in a more effective manner when compared to common VM-based traffic generation setups.
4. We propose DetGen, a container-based traffic generation paradigm that provides accurate control and labels over traffic microstructures, and experimentally demonstrate the level of provided generative determinism to traditional generation set-ups.

DetGen and the DetGen-IDS data are openly accessible for researchers on *GitHub*.

1.1 Outline

The remainder of the paper is organized as follows. Section 2 discusses the need for generating probing data with sufficient microstructure control before presenting the probing and corresponding improvement of a state-of-the-art intrusion detection model as a motivating example. Section 3 discusses how to generate probing data appropriately for pretained models, and provides an overview over the DetGen-IDS data. Section 4 proceeds to examine over which traffic characteristics DetGen exerts control and the corresponding control level. Section 5 provides details over the design paradigm of DetGen and the resulting advantages over traditional setups, while Section 6 discusses the level of control DetGen provides when compared to traditional setups. Section 7 concludes our work.

1.2 Existing datasets and corresponding ground-truth information

Real-world NID-datasets such as those from the Los Alamos National Laboratory [11] (LANL) or the University of Grenada [16] provide large amounts of data from a particular network in the form of flow records. Due to the lack of monitoring and traffic anonymisation, it is impossible for researchers to extract detailed information about the specific computational activity associated with a particular traffic sample. Synthetic NID-datasets such as the CICIDS-17 and 18 [26] or the UNSW-NB-15 [20] aim to provide traffic from a wide range of attacks as well as an enterprise-like topology in the form of `pcap`-files and flow-statistics. The CICIDS-17 data for example contains 5 days of network traffic from 12 host that include different Windows, Ubuntu, and Web-Server versions, and covers attacks from probing and DoS to smaller SQL-injections and infiltrations. While some effort is put in the generation of benign activities using activity scripting or traffic generators, we have seen no attention being spent at monitoring

these activities accordingly, which leaves researchers with the limited information available through packet inspection. Furthermore, synthetic datasets can be criticised for their limited activity range, such as the CICIDS-17 dataset where more than 95% of FTP-transfers consist downloading the Wikipedia page for 'Encryption' [24], which leads to insufficient structural nuances to for effective probing.

1.3 Scope of DetGen

The scope of DetGen is to generate traffic with near-deterministic control over factors that influence microscopic packet- and flow-level structures. DetGen separates program executions and traffic capture into distinct containerised environments to exclude any background traffic events, simulates influence factors such as network congestion, communication failures, data transfer size, content caching, or application implementation.

2 Methodology and example

Assume the following problem: You are designing a packet-level traffic classifier which is generating a significant number of false positives, something that is still a common problem for the state-of-the-art [22]. The false positives turn out to be caused by a particular characteristic such as unsuccessful logins or frequent connection restarts. However, existing real-world or synthetic datasets do not contain the necessary information to associate traffic events with these characteristics, which prevents you from identifying the misclassification cause effectively. To address this problem, we need a way to controllably generate and label traffic microstructures caused by these characteristics.

To provide an example, we look at a *Long-Short-Term Memory* (LSTM) network, a deep learning design for sequential data, by Hwang et al. [8], which is designed to classify attacks in web traffic and has achieved some of the highest detection rates of packet-based classifiers in a recent survey [29]. Through probing we will learn that retransmissions in a packet sequence dramatically deplete the model's classification accuracy. We take the following steps:

Step 1: Determine model performance and feed it suitable probing traffic.
Step 2: Examine the correlation between traffic misclassification scores and the generated traffic microstructure labels to find a likely cause.
Step 3: Examine at which latency levels specific connections are misclassified.
Step 4: Generate two similar connections, with one exposed to strong packet latency.
Step 5: Show that by removing retransmission sequences in the pre-processing, misclassification is significantly reduced.

Step 1: To detect SQL injections, we train the model on the CICIDS-17 dataset [26] (85% of connections). For the evaluation, we also include a set of

HTTP-activities generated by DetGen (7.5%) that mirror the characteristics in the training data, as explained in Section 3. In total, we use 30,000 connections for training and for evaluating the model, or slightly under 2 million packets. The initially trained model performs relatively well, with an *Area under curve* (AUC)-score of **0.981**, or a detection/false positive rate[1] of **96%** and **2.7%**. However, to enable operational deployment the false positive rate would need to be several magnitudes lower [18].

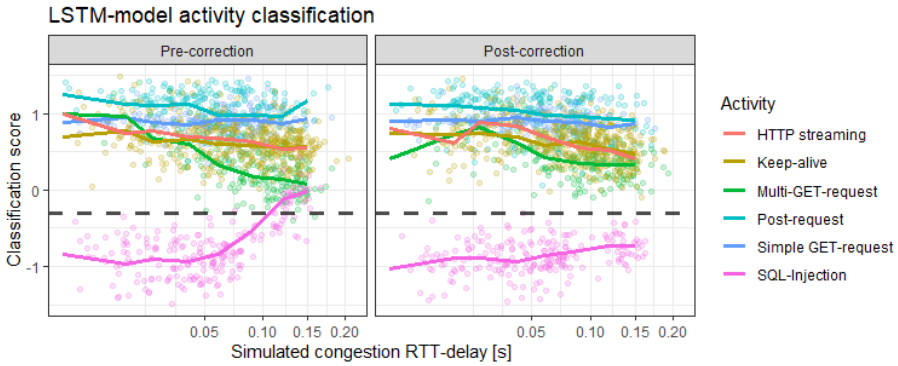

Fig. 2: Scores for the LSTM-traffic model before and after the model correction.

Step 2: Now suppose we want to improve these rates to both detect more SQL-injections and retain a lower false positive rate. To start, we explore which type of connections are misclassified most often. We retrieve the classification scores for all connections and measure their linear correlation to the microstructure labels available for the probing data. The highest misclassification ratio was measured for one of the three SQL injection scenarios (19% correlation) and connections with multiple GET-requests (11% correlation). When not distinguishing activities, we measured a high misclassification correlation with simulated packet latency (12%), which we now examine. More details on this exact procedure can be found in (citation currently blinded).

Step 3: Fig. 2 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, while the right panel depicts scores after the model correction that we introduce further down. The left panel shows that classification scores are well separated for lower congestion, but 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

[1] tuned for the geometric mean

corresponding retransmission attempts. Both of these factors can decrease 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.

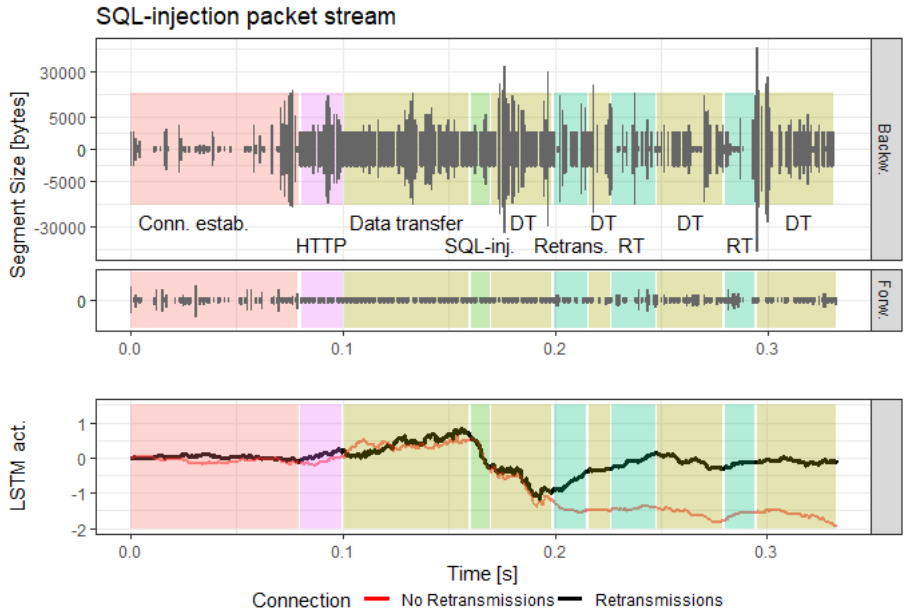

Fig. 3: LSTM-output activation in dependence of connection phases.

Step 4: We use DetGen to generate two similar connections, where one connection is subject to moderate packet latency and corresponding reordering while the other is not. DetGen's ability to shape traffic in a controlled and deterministic manner allows us to examine the effect of retransmission sequences on the model output and isolate it from other potential influence factors. Fig. 3 depicts the evolution of the LSTM-output layer activation in dependence of difference connection phases for the connection subject to retransmissions. 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-output activation while processing the packet streams. The red line shows the output for the connection without retransmissions[2] as a comparison. 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, just as it is supposed to.

[2] scaled temporally to the same connection phases

The correct 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 and return to the correct output activation. When we compare this to the connection without retransmissions, depicted as the red line in Fig. 3, we do not encounter this depletion effect. Instead, the negative activation persists after the injection phase.

Step 5: Based on this analysis, we try to correct the existing model with a simple fix by excluding retransmission sequences at the pre-processing stage. This leads to significantly better classification results during network latency, as visible in the right panel of Fig. 2. SQL-injection scores are now far-less 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 improved to **99.1%** and false positives to **0.345%**, a five-fold improvement from the previous false positive rate of 2.7%.

3 Reconstructing an IDS-dataset for efficient probing

Moving towards a more general dataset constructed to apply this probing methodology, we constructed *DetGen-IDS*. This dataset is suitable to quickly probe ML-model behaviour that were trained on the CICIDS-17 dataset [26]. The dataset mirrors properties of the CICIDS-17 data to allow pre-trained models to be probed without retraining. The *DetGen-IDS* data therefore serves as complementary probing data that provides microstructure labels and a sufficient and controlled diversity of several traffic characteristics that is not found in the CICIDS-17 data.

We focus on mirroring the following properties from the CICIDS-17 data:

1. **Application layer protocols (ALP)**
2. **ALP implementations**
3. **Typical data volume for specific ALPs**
4. **Conducted attack types**

Extracting more information on characteristics such as conducted activities of current NID-datasets is difficult for the reasons explained in 1.2. However, our examination shows that aligning these high-level features with the original training data helps to significantly reduce the validation error of a model on the probing data.

We then took the following steps to extract the necessary information from the CICIDS-17 data and implement the traffic-generation process accordingly:

1. The primary ALPs in the dataset can be identified using their corresponding network ports. We ordered connections by the frequency of their respective port, and excluded connections that do not transmit more than 15 packets per connection as these do not provide enough structure to create probing data from it. This leaves us with the ALPs *HTTP/SSL, SMTP, FTP, SSH, SQL, SMB, and NTP*. We had already implemented traffic scenarios for each of them except SMB and LDAP, which we then added to the catalogue described in Section

5.3. Table 1 displays the frequency of the most common ALPs in the CICIDS-17 along with their average size and packet number per connection and how we adopted them in the DetGen-IDS data.

2. Most of the used ALP implementations, such as *Apache* and *Ubuntu Webserver* for HTTP, could be gathered from the description of the CICIDS-17 dataset. When this was not the case, it is mostly possible to gather this information by inspecting a few negotiation packets for the corresponding ALP with Wireshark to identify the TLS version or the *OpenSSH*-client. The correct ALP implementation can then be included in the traffic generation process by simply identifying and including a Docker-container that matches the requirements, which is explained more in Section 5.3.

		CICIDS-17			DetGen-IDS	
ALP	Port	Av. Conn. Size	Av. Packets /Conn.	# Packets	# Packets	# Activities
HTTP	80	131626.4	120.4	26631853	724032	7
HTTPS	443	24637.5	36.7	18531661	432104	7
DNS	53	286.2	3.6	3515510	-	-
SSH	22	4699.6	40.9	430380	379421	13
LDAP	389	5429.2	22.3	133471	94587	3
FTP	21	311.3	41.7	121472	183587	9
NetBIOS	137	773.6	14.3	111341	-	-
SMB	445	12941.5	61.9	88175	47945	3
NTP	123	157.0	3.2	73057	1243	1
SMTP	465	2663.5	21.5	77650	104967	3
Kerberos	88	2687.7	6.9	38262	-	-
mDNS	5353	3685.5	35.5	24592	-	-

Table 1: Common ALPs in CICIDS-17 data

3. Since the total size of a connection is one of the most significant features for its classification, we restrict connections in the DetGen-IDS data to cover the same range as their counterparts in the CICIDS-17 data. For this, we extracted the maximum and minimum connection size for each ALP in the benign data and use it as a cut-off to remove all connections from the DetGen-IDS data that do not meet this requirement.

4. Included attacks are well documented in the CICIDS-17 description. These include *SQL-injections, SSH-brute-force, XSS, Botnet, Heartbleed, GoldenEye, and SlowLoris*. We aim to cover as many of these attack types in the DetGen-IDS data as well as adding them to the overall DetGen-attack-catalogue. 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.

In addition to the `pcap`-files, we used the *CICFlowMeter* to generate the same 83 flow-features as included in the CICIDs-17 data. Table to be added displays the content and statistics of the DetGen-IDS data.

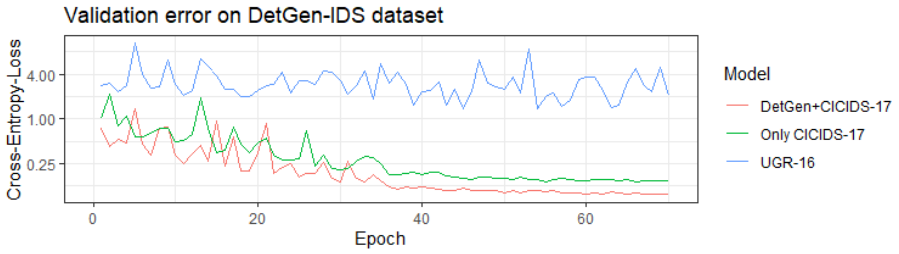

Fig. 4: Validation errors of LSTM-model [3] on DetGen-IDS data.

In Fig. 4, we compare the validation error of a recent LSTM-model for network intrusion detection by Clausen et al. [3] on the DetGen-IDS data to demonstrate that a model trained on the CICIDS-17 data is able to perform well without retraining. We distinguish models when trained exclusively on the CICIDS data (green), and when also trained on the probing data (red). 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 [16], blue). This does not fully prove that every model is able to transfer observed structures between the two datasets, but it gives an indicator that they mirror characteristics.

4 Traffic microstructures and their influence factors

The biggest and most obvious influence on traffic microstructures is the choice of the application layer protocols. For this reason, the range of protocols is often used as a measure for the diversity of a dataset. However, while the attention to microstructures in current NID-datasets stops here, computer communication involves a myriad of other different computational aspects that shape observable traffic microstructures. Here, we highlight and quantify the most dominant ones, which will act as a justification for the design choices we outline in Section 5.4. We look at both findings from previous work as well as our own experimental results.

1. Performed task and application. The conducted computational task as well as the corresponding application ultimately drives the communication between computers, and thus hugely influences characteristics such as the direction of data transfer, the duration and packet rate, as well as the number of connections established. These features are correspondingly used extensively in application fingerprinting, such as by Yen et al. [33] or Stober et al. [28].

2. Application layer implementations. Different implementations for TLS, HTTP, etc. can yield different computational performance and can perform handshakes

differently and differ in multiplexing channel prioritisation, which can significantly impact IAT times and the overall duration of the transfer, as shown in a study by Marx et al. [17] for the QUIC/HTTP3 protocol[3].

3. LAN and WAN congestion. Low available bandwidth, long RTTs, or packet loss can have a significant effect on TCP congestion control mechanisms that influence frame-sizes, IATs, window sizes, and the overall temporal characteristic of the sequence, which in turn can influence detection performance significantly as shown in Section 2.

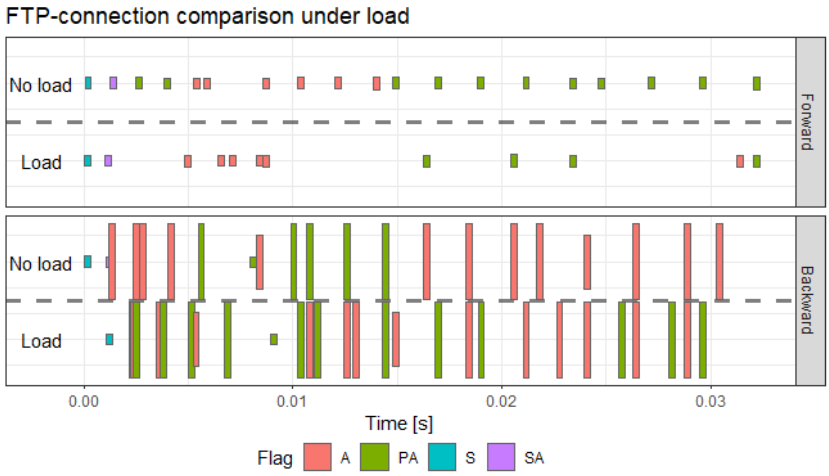

Fig. 5: Packet-sequence similarity comparison under different load.

4. 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 a FTP-client sends significantly fewer `PUSH`-packets when under heavy computational load. Colours indicate packet flags while the height of the packets indicates their size. This effect is dependent on the application layer protocol, where at a load number of 3.5 we see about 60% less upstream data-packets while the downstream is only reduced by 10%, compared to HTTP where both downstream and upstream packet rates are throttled by about 40%.

5. 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 and less

[3] Fig. 2 in [17] illustrates these differences in a nice way

connections being established. For caching, this can result in less than 10% of packets being transferred, as shown by Fricker et al. [4].

6. User and background activities. The choice and usage frequency of an application and task by a user, sometimes called *Pattern-of-Life*, governs the larger-scale temporal characteristic of a traffic capture, but also influences the rate and type of connections observed in a particular time-window [1]. The mixing of different activities in a particular time-window can severely impact detection results of recent sequential connection-models, such as by Radford et al. [23] or by Clausen et al. [3]. To quantify this effect, we look at FTP-traffic in the CICIDS-17 dataset. As explained in Section 1.2, the FTP-traffic overwhelmingly corresponds to the exact same isolated task, and should therefore spawn the same number of connections in a particular time window. However, we observe additional connections from other activities within a 5-second window for 68% of all FTP-connections, such as depicted in Table 2, which contains FTP-, HTTPS- and DNS-, as well as additional unknown activity.

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 2: 5-second window for host 192.168.10.9 in the CICIDS-17 dataset.

Other prominent factors that we found had less effect on traffic microstructures include:

7. 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 [25]. In practice, this effect seems to be constrained to large WAN-servers and routers. When varying the stack load in otherwise constant settings on an Ubuntu-host, we did not find any notable effect on packet sequences when comparing the corresponding traffic with a set of three similarity metrics. More details on this setting and the metrics can be found in Section 6.

8. Network configurations. Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes, and have been exploited in IP fragmentation attacks. However, these settings have been standardised for most networks, as documented in the CAIDA traffic traces [31].

We designed DetGen to control and monitor factors 1-6 to let researchers explore their impact on their traffic models, while omitting factors 7 and 8 for the stated reasons.

5 DetGen: precisely controlled data generation

5.1 Design overview

Detgen is a container-based network traffic generation framework that distinguishes itself by providing precise control over various traffic characteristics and providing extensive ground-truth information about the traffic origin. In contrast to the pool of programs running in a VM-setup, such as used in the generation of the CICIDS-17 and 18 [26], or UGR-16 [16], DetGen separates program executions and traffic capture into distinct containerised environments in order to shield the generated traffic from external influences.Traffic is generated from a set of scripted *scenarios* that define the involved devices and applications and strictly control corresponding influence factors. Fig. 6 provides a comparison of the DetGen-setup and traditional VM-based setups and highlights how control and monitoring is exerted.

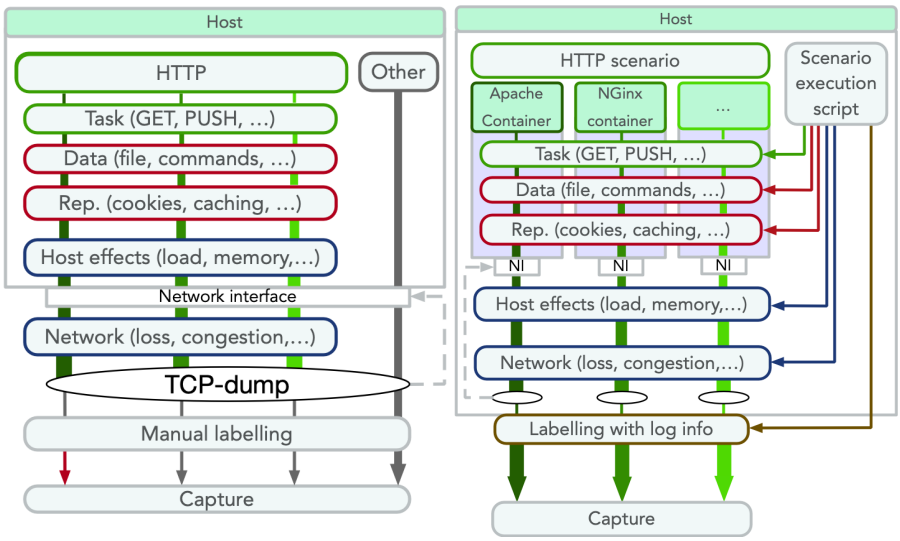

Fig. 6: Traditional traffic-generation-setups (left), and DetGen (right).

5.2 Containerization and activity isolation

As we will demonstrate in Section 6, 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 and generate traffic. The corresponding one-to-one correlation between processes and network traces allows us to capture traffic directly from the process and produce labelled datasets with granular ground truth information.

Additionally, 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, allowing a certain level of **determinism** and reproducibility in the data generation.

5.3 Activity generation

Scenario. We define a *scenario* as a composition of containers conducting a specific interaction. Each scenario produces traffic from a setting with two (client/server) or more containers, with traffic being captured from each container's perspective. This constructs network datasets with total interaction capture, as described by Shiravi et al. [27]. Examples may include an FTP interaction, an online login form paired with an SQL database, or a C&C server communicating with an open backdoor. Our framework is modular, so that individual scenarios are configured, stored, and launched independently. We provide a complete list of implemented scenarios in Table 3 in the Appendix.

Task. 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. To explore the breadth of the corresponding traffic structures efficiently, we prioritise tasks that cover aspects such as the 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).

5.4 Simulation of external influence

Caching/Cookies/Known server. 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.

Network effects. Communication between containers takes place over a virtual bridge network, which provides far higher and more reliable throughput than in real-world networks [5]. To retard and control the network reliability and congestion to a realistic level, we rely on *NetEm*, an enhancement of the Linux

traffic control facilities for emulating properties of wide area networks from a selected network interface [7].

We apply NetEm 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 are drawn from a binomial distribution, which has been found to emulate real-world settings well [9]. 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 Data generation

Execution script. DetGen generates traffic through executing execution script that are specific to the 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.

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 execution script then stores all parameters (conducted task, mean packet delay, ...) and descriptive values (input file size, communication failure, ...) for the chosen settings.

6 Traffic control and generative determinism of DetGen

We now assess the claim that DetGen controls the outlined traffic influence factors sufficiently, 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 three traffic types, namely HTTP, file-synchronisation, and botnet-C&C, each in four configurations that varied in terms of conducted activity, data/credentials as well as the applied load and congestion. Within each configuration all controllable factors are held constant

to test the experimental determinism and reproducibility of DetGen's generative abilities. 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, such as depicted in Fig. 6. Such a setup is for example used in the generation of the UGR-16 data [16].

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:

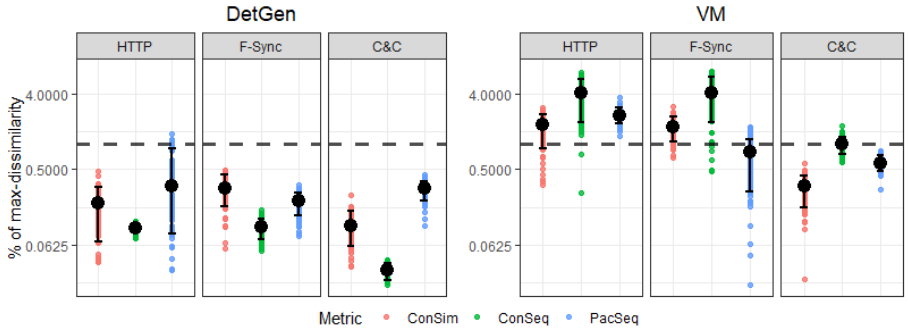

Fig. 7: Dissimilarity scores for DetGen and a regular VM-setup, on a log-scale.

- **Overall connection similarity** We use the 82 flow summary statistics (IAT and packet size, TCP window sizes, flag occurrences, burst and idle periods) provided by CICFlowMeter [13], and measure the cosine similarity between connections, which is also used in general traffic classification [2].
- **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, as used by Yen et al. [33] 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 handshakes etc., we use a Markovian probability matrix for packet flags, IATs, sizes, and direction conditional on the previous packet. We do this for sequences of 15 packets and use the average sequence likelihood as this accommodates better for marginal shifts in the sequence.

We normalise all dissimilarity scores by dividing them by the maximum dissimilarity score measured for each traffic type to put the scores into context. For each configuration, we generate 100 traffic samples and apply the described dissimilarity measures to 100 randomly drawn sample pairs. Fig. 7 depicts the resulting dissimilarity scores on a log-scale.

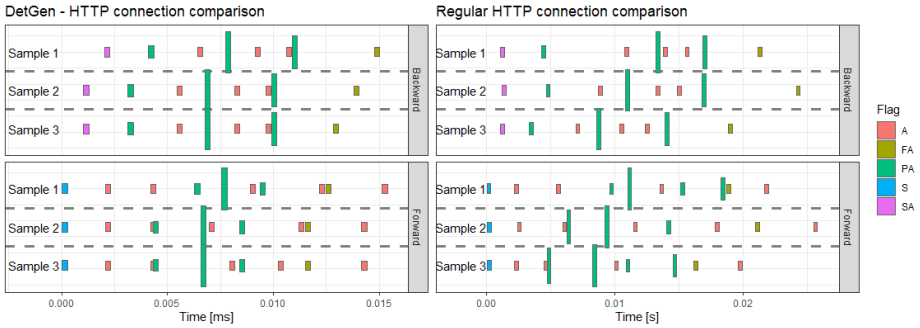

Fig. 8: Packet-sequence similarity comparison for HTTP-activity for DetGen and a regular setting.

The DetGen-scores yield consistently less than 1% of the dissimilarity observed on average for each activity. Scores are especially low when compared to traffic groups collected in the VM setting, which are consistently more dissimilar, in particular for connection-sequence metrics, where the average dissimilarity is more than 30 times higher than for the DetGen setting. Manual inspection of the VM-capture showed that high dissimilarity is caused by additional flow events from background activity (OS and application HTTP, NTP, DNS, device discovery) being present in about 24% of all captures. . 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. 8 depicts an exemplary comparison between HTTP-samples generated using DetGen versus generation using the VM-setup. Colours indicate packet flags while the height of the packets indicates their size. Even though samples from DetGen are not perfectly similar, packets from the VM-setup are subject to more timing perturbations and reordering as well as containing additional packets. Additionally, the packet sizes vary more in the regular setting.

These results confirm that DetGen exerts a high level of control over traffic shaping factors while providing sufficient determinism to guarantee ground-truth traffic information.

7 Conclusions

In this paper, we described and examined a tool for generating traffic with controllable and extensively labelled traffic microstructures with the purpose of probing machine-learning-based traffic models. For this, we demonstrated the impact that probing with carefully crafted traffic microstructures can have for improving a model with a state-of-the-art LSTM-traffic-classifier with a detection rate that improved by more than 3% after understanding how the model processes excessive network congestion.

To verify DetGen's ability to control and monitor traffic microstructures, we performed experiments in which we quantified the experimental determinism of DetGen and compared it to traditional VM-based capture setups. Our similarity metrics indicate that traffic generated by DetGen is on average 10 times, and for connection sequences up to 30 times more consistent.

Alongside this work, we are releasing DetGen-IDS, a substantial dataset suitable for probing models trained on the CICIDS-17 dataset. This data should make it easier for researchers to understand where their model fails and what traffic characteristics are responsible to subsequently improve their design accordingly.

DetGen and the corresponding dataset are openly accessible for researchers on GitHub.

Difficulties and limitations: While the control of traffic microstructures helps to understand packet- or connection-level models, it does not replicate realistic network-wide temporal structures. Other datasets such as UGR-16 [16] or LANL-15 [30] are currently better suited to examine models of large-scale traffic structures.

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 alter the actual detection performance of a model.

The artificial randomisation of traffic shaping factors can currently not completely generate real-world traffic diversity. This problem is however more pronounced in commonly used synthetic datasets such as CICIDS-17, where for example most FTP-transfers consist of a client downloading the same text file.

Discussions about the implications of the model correction proposed in Section 2 are above the scope of this paper, and there likely exist more complex and suitable solutions.

Future work: DetGen is currently only offering insufficient control over underlying **application-layer implementations** such as TLS 1.3 vs 1.2. In theory, it should be unproblematic to provide containers with different implementations, and we are currently investigating how to compile containers in a suitable manner.

We are currently investigating how to better simulate causality in connection spawning and other **medium-term temporal dependencies**, such as by importing externally generated activity timelines from tools such as Doppelganger [14].

A project we are currently working on is to embed traffic scenarios into a larger and more complex **network topology** using MiniNet [12].

References

1. F. J. Aparicio-Navarro, J. A. Chambers, K. Kyriakopoulos, Y. Gong, and D. Parish. Using the pattern-of-life in networks to improve the effectiveness of intrusion detec-

tion systems. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2017.
2. Y. Aun, S. Manickam, and S. Karuppayah. A review on features' robustness in high diversity mobile traffic classifications. *International journal of communication networks and information security*, 9(2):294, 2017.
3. H. Clausen, G. Grov, M. Sabaté, and D. Aspinall. Better anomaly detection for access attacks using deep bidirectional LSTMs. In *International Conference on Machine Learning for Networking*, pages 1–14. Springer, 2020.
4. 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.
5. M. Gates and A. Warshavsky. Iperf Man Page. `https://linux.die.net/man/1/iperf`. Accessed: 2019-08-11.
6. A. Haque, M. Guo, P. Verma, and L. Fei-Fei. Audio-linguistic embeddings for spoken sentences. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7355–7359. IEEE, 2019.
7. S. Hemminger et al. Network emulation with NetEm. In *Linux conf au*, pages 18–23, 2005.
8. 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.
9. 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.
10. H. Kamper, Y. Matusevych, and S. Goldwater. Multilingual acoustic word embedding models for processing zero-resource languages. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6414–6418. IEEE, 2020.
11. A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.
12. B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.
13. A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani. Characterization of tor traffic using time based features. In *ICISSp*, pages 253–262, 2017.
14. Z. Lin, A. Jain, C. Wang, G. Fanti, and V. Sekar. Generating high-fidelity, synthetic time series datasets with doppelganger. *arXiv preprint arXiv:1909.13403*, 2019.
15. S. R. Livingstone and F. A. Russo. The ryerson audio-visual database of emotional speech and song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in north american english. *PloS one*, 13(5):e0196391, 2018.
16. G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón. UGR '16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Computers & Security*, 73:411–424, 2018.
17. 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.
18. P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman. An overview of issues in testing intrusion detection systems, 2003.

19. Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.
20. N. Moustafa and J. Slay. UNSW-NB15: a comprehensive data set for network intrusion detection systems. In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.
21. M. Nasr, A. Bahramali, and A. Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.
22. A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos. From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials*, 2018.
23. B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson. Network traffic anomaly detection using recurrent neural networks. *arXiv preprint arXiv:1803.10769*, 2018.
24. M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019.
25. 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.
26. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.
27. 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.
28. 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.
29. 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.
30. M. J. M. Turcotte, A. D. Kent, and C. Hash. Unified Host and Network Data Set. *ArXiv e-prints*, Aug. 2017.
31. C. Walsworth, E. Aben, K. Claffy, and D. Andersen. The CAIDA UCSD anonymized internet traces 2018," , 2018.
32. 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.
33. 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.

A Existing Scenarios

DetGen contains 31 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. According to our evaluation, our scenarios can generate datasets containing the protocols that make up at least 87.8% (MAWI), 98.3% (CICIDS 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	9
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
SMB	File-sharing	3
LDAP	Access directory services	3

Name	Description	#Ssc.
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 3: Currently implemented traffic scenarios along with the number of implemented subscenarios

In total, we produced 19 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 behaviour. 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 behaviour or Cryptojacking previously had no available datasets for study despite need from academic and industrial researchers.