Controlling traffic microstructures for model probing using DetGen

No Author Given

No Institute Given

Abstract. In this paper, we argue that ground-truth insights and control over fine-grained traffic microstructures are crucial to understand and improve NID-model behaviour, but currently overlooked in benchmark datasets. We demonstrate how probing of a traffic classifier with appropriately generated and labelled microstructures enables the association of misclassification with particular traffic characteristics and correspondingly reduce the false-positive rate by more than 500%. We examine how large the impact of specific factors is on traffic microstructures, and how well their influence in the traffic generation process can be controlled to provide ground-truth information. We then demonstrate how to generate traffic appropriately to probe pre-trained NID-models. For this, we introduce DetGen, a traffic generation tool that provides microstructure control and corresponding ground-truth labels.

1 Introduction

In this work, we examine which factors shape traffic structures on a microscopic level, and how controlling and monitoring these factors improves the probing of data-driven network intrusion detection (NID) models. For this, we perform several experiments to demonstrate how to produce traffic effectively on a state-of-the-art traffic classifier, and present DetGen, a traffic generation tool that provides near-deterministic control over various **microstructure**-shaping factors such as conducted activity, communication failures or network congestion along with corresponding ground-truth labels. Our motivation stems from the lack of attention that current benchmark datasets spent on these factors.

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 in other domains such as the *What-If tool* [30] underline the importance of model validation, but are not suitable providing traffic probing data.

Machine-learning breakthroughs in many other fields have often been reliant on a precise understanding of data structure and corresponding descriptive labelling to develop more suitable models. In *automatic speech recognition (ASR)*,

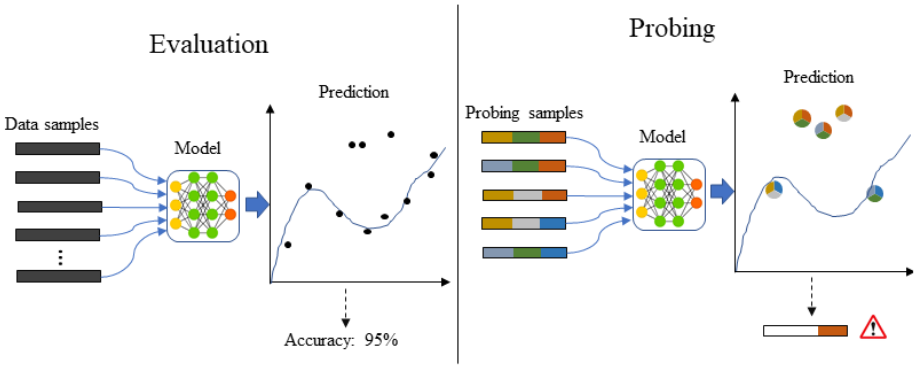


Fig. 1: Model evaluation and model probing with controlled data characteristics.

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 identify if their model is susceptible to structural misclassification, but also inspire new methods to capture and understand these implicit structures [7], which in turn leads to design improvements of general speech recognition models [11].

Similarly, several approaches to enhance the way information is collected and presented to improve understanding between data and cyber-detection systems. Researchers today use threat reports to enhance behavioural malware labelling [26], monitor cloud applications from a hypervisor-level using VM-introspection [4], and develop data provenance tools to expand the information contained in system execution logs [2].

For network traffic generation and monitoring, the current quasi-benchmark datasets pay more attention to the inclusion of a wide variety of attacks rather than the control and documentation of the generated traffic. This situation has so far lead researchers to predominantly apply a number of ML-models to traffic datasets in the hope of edging out competitors, without performing any model validation or improvement through probing and corresponding failure analysis.

This work provides the following contributions:

1. We demonstrate how to generate probing traffic that is suitable to probe models pre-trained on a NIDS-dataset.
2. We probe a state-of-the-art LSTM traffic classifier with this traffic dataset, and demonstrate how to lower false-positives effectively by understanding where the model fails to process particular traffic structures correctly.

3. We examine experimentally how different factors affect traffic microstructures, and how well they can be controlled effectively when compared to common VM-based traffic generation setups.
4. We describe how the container-based design paradigm of DetGen provides accurate control over traffic characteristics as well as corresponding ground truth information, and compare the design advantages 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 purpose and necessity 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 [12] (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 [24] or the UNSW-NB-15 [19] 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' [22], which leads to insufficient structural nuances to for effective probing.

2 Motivation and Methodology

2.1 Improving a classifier with traffic control and microstructure labels

Assume the following problem: You are designing a packet-level traffic classifier which is generating a significant amount of false-positives, something that is still a common problem for the state-of-the-art [20]. 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. We designed DetGen to generate traffic with sufficient ground-truth information to allow effective association of such model failures with traffic 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. [9], which is designed to classify attacks in web traffic and achieved some of the highest detection rates of packet-based classifiers in a recent survey [28]. Through probing we will learn tha retransmissions in a packet sequence deplete the models classification accuracy. We will take the following steps:

- We generate suitable probing traffic and input it to the trained model.
- We examine the correlation between traffic misclassification scores and the generated traffic microstructure labels, which is strongest for simulated packet reordering.
- We generate two similar connections, with one exposed to strong packet loss to examine how it affects the processing.
- We then show that by removing retransmission sequences in the preprocessing, misclassification is significantly reduced.

To detect SQL-injections, we train the model on the CICIDS-17 dataset [24] (85% of connections). 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 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)-scoreof **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].

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 perform a correlation analysis between the additional microstructure labels available for the probing data, and the classification score. The highest misclassification ratio was measured for one of the three SQL injection scenarios (19% correlation) and connections with

[1] tuned for the geometric mean

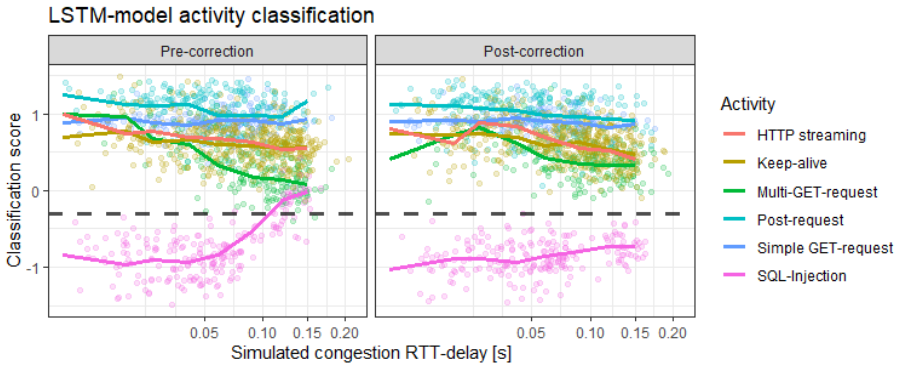


Fig. 2: Scores for the LSTM-traffic model before and after the model correction.

multiple GET-requests (11% correlation). When not considering particular activities, we measured a high misclassification correlation with simulated packet latency (12%), which we now examine.

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 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.

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 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

[2] scaled temporally to the same connection phases

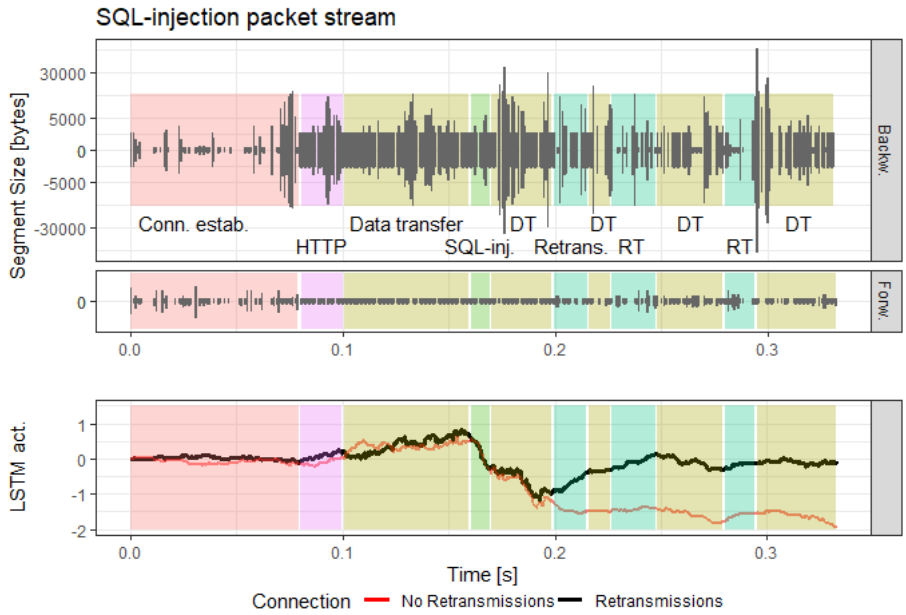


Fig. 3: LSTM-output activation in dependence of connection phases.

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.

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.

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 and false-positives improved to **99.1%** and **0.345%**. Discussions about the implications of this correction is above the scope of this paper, and there likely exist more more complex and suitable solutions.

2.2 Scope of DetGen

DetGen generates traffic 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- or individual flow-level.

DetGen separates program executions and traffic capture into distinct containerised environments in order to exclude any background traffic events, and therefore provides 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 6, and consequently provide better reproducible data. In order to examine additional effects on traffic microstructures, DetGen simulates influence factors such as network congestion, communication failures, data transfer size, content caching, or application implementation.

3 Reconstructing an IDS-dataset for efficient probing

We constructed a dataset, called *DetGen-IDS*, that is suitable to quickly probe ML-model behaviour that were trained on the CICIDS-17 dataset [24], which we used for the probing example in Section 2.1. The dataset mirrors properties of the CICIDS-17 data to allow pre-trained models to be probed without the need to retrain the model on the probing data. 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 necessarily found in the CICIDS-17 data.

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

- **Application layer protocols (ALP)**
- **ALP implementations**
- **Typical data volume for specific ALPs**
- **Conducted attack types**

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

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 4 packets per exchange 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, which we then added to the catalogue described in Section 5.3. Table in Appendix, to be added displays the corresponding frequency of protocols.

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. In cases where this was not the case, we inspected 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.

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.

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.

These steps can also be applied for other synthetic datasets such as UNSW-NB15 that have sufficient information available in their documentation or the `pcap`-capture. Since DetGen already covered ALPs that are responsible for 98% of connections in the CICIDS-17 data, it was most straightforward for us to generate probing data for the CICIDS-17 data.

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.

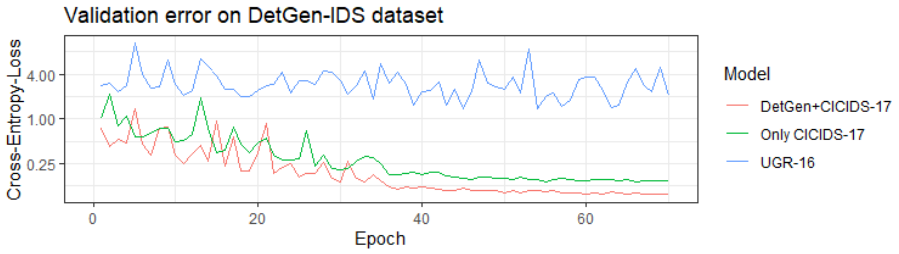


Fig. 4: Validation errors of LSTM-model [3] on DetGen-IDS data.

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 proof that every model is able to transfer observed structures between the two datasets, but it gives an indicator that they mirror characteristics.

ID	Activity	kn.-host	...	File-l. [b]	load	lat. [s]
ABC	F.-transfer with passw.	Yes	...	256236	1.01	14
DEF	Keyscanning	No	...	1162	2.39	248
⋮						

Table 1: Exemplary SSH-labels

To be added

4 Traffic microstructures and their influence factors

Without doubt the biggest and most obvious impact on the captured 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, computer communication involves a myriad of different computational aspects, which are mostly overlooked in the data generation process of existing NIDS-datasets. 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. [31] or Stober et al. [27].

2. Application layer implementations Different implementations for TLS, HTTP, etc. can yield different computational performance and can perform handshakes different 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] Fig. 2 in [17] illustrates these differences in a nice way

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 2.1.

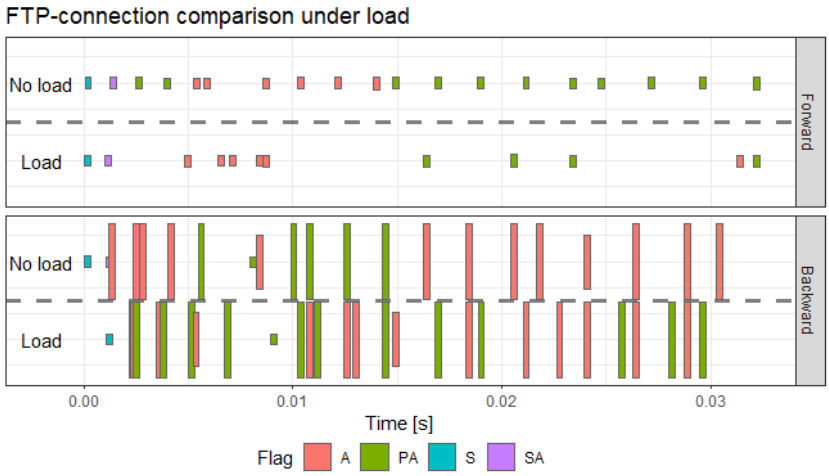


Fig. 5: Packet-sequence similarity comparison under different load.

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 connections being established. For caching, this can result in less than 10% of packets being transferred, as shown by Fricker et al. [5].

6. User and background activities The choice and usage frequency of an application and task by a user 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. 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. [21] 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 [23]. 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.

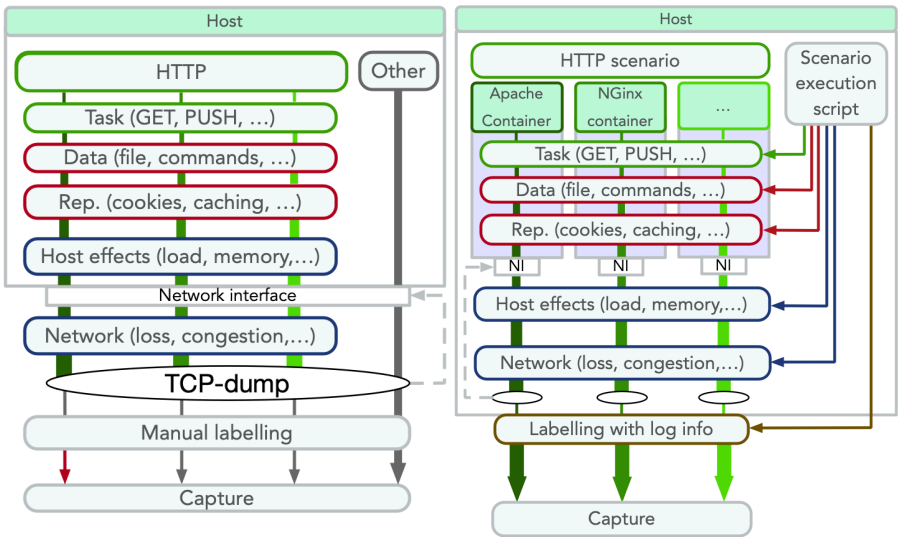
8. Network configurations Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes, but are standardised for most networks, as documented in the CAIDA traffic traces [29].

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 Architecture

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, 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* that define the involved devices and applications and strictly control corresponding influence factors. Fig. 6 provides a technical design comparison of the DetGen-setup and traditional VM-based setups and highlights how control and monitoring is exerted.



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

Fig. 6: Comparison of traditional traffic-generation-setups (a), and DetGen (b).

5.2 Containerization and activity isolation

As we demonstrated 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.

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. [25]. 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.

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. To explore the breadth of the corresponding traffic structures efficiently, we prioritise to add 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 [6]. 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 [8].

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 is drawn from a binomial distribution, which has been found to emulate real-world settings well [10]. 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.

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 generation 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. Within each configuration all controllable influence 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. 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:

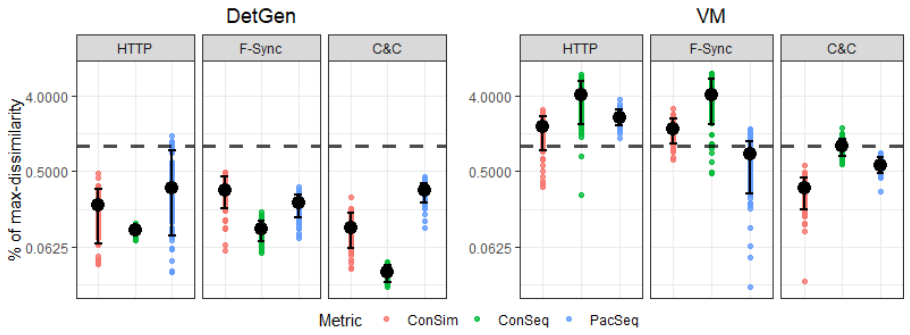


Fig. 7: Comparison of dissimilarity scores for DetGen and a regular VM-setup, on a log-scale.

- **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 [1].
- **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. [31] 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 of each group as a similarity measure.

We normalise all dissimilarity scores by dividing them by the maximum dissimilarity score measured for each traffic type so that the reader can put the scores into context. For each of the configurations, we generate 100 traffic samples and apply the described dissimilarity measures to 100 randomly drawn pairs sample pairs. Fig. 7 depicts the calculated dissimilarity scores for DetGen and the VM-setup.

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. This is likely caused by additional background traffic 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. 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.

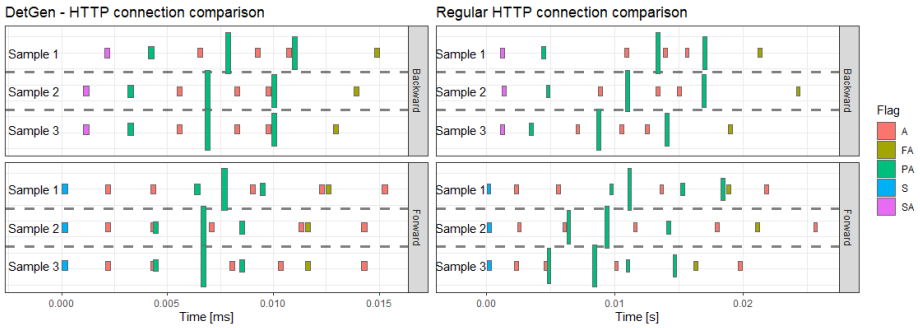


Fig. 8: Packet-sequence similarity comparison for HTTP-activity for DetGen and a regular setting.

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 similar-

ity metrics indicate that traffic generated by DetGen is up to 30 times more consistent.

We also release 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 in order to subsequently improve their design accordingly.

DetGen and the corresponding dataset are openly accessible for researchers.

7.1 Difficulties and limitations

While the control of traffic microstructures 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 [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'.

7.2 Future work

We paid meticulous attention to enable control over as many traffic impact factors as possible. However, 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 for each scenario to provide this control. However we faced difficulties to compile containers in a suitable manner and are currently investigating, how to improve DetGen on this shortcoming.

In addition to that, we are currently investigating how to better simulate causality in connection spawning and other **medium-term temporal dependencies**. One idea is to import externally generated activity timelines using tools such as Doppelganger [14] for which the corresponding traffic is generated to the corresponding time-stamps.

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

References

1. 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.
2. 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.
3. H. Clausen, G. Grov, M. Sabaté, and D. Aspinall. Better anomaly detection for access attacksusing deep bidirectional lstms. In *International Conference on Machine Learning for Networking*, pages 1–14. Springer, 2020.
4. B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *2011 IEEE symposium on security and privacy*, pages 297–312. IEEE, 2011.
5. 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.
6. M. Gates and A. Warshavsky. Iperf Man Page. `https://linux.die.net/man/1/iperf`. Accessed: 2019-08-11.
7. 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.
8. S. Hemminger et al. Network emulation with netem. In *Linux conf au*, pages 18–23, 2005.
9. 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.
10. 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.
11. 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.
12. A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.
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. N. Moustafa and J. Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.
29. C. Walsworth, E. Aben, K. Claffy, and D. Andersen. The caida ucsd anonymized internet traces 2018,", 2018.
30. 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.
31. 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

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% (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	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

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 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 3.

B CICIDS-17 statistics