Contribution Title

No Author Given

No Institute Given

Abstract. We introduce DetGen, a tool that generates traffic to improve the ability to probe and understand model behaviour in data-driven network intrusion detection, and help explain the corresponding decisions made by a model. DetGen operates under a new design paradigm based on containerisation and reproducibility in order to closely control different factors that influence generated network traffic and provide cross-linkage information between captured traffic and these factors. We demonstrate the effectiveness of probing ML-models with controllable traffic samples, and perform an analysis of the level of control DetGen provides. We furthermore release a dataset suitable for rapidly probing pre-trained models.

1 Introduction

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

Data-driven traffic analysis and attack detection is a centrepiece of network intrusion detection research, and the idea of training detection systems on large amounts of network traffic appears like the solution to cyber-threats, and has received tremendous attention in the academic literature. Despite this, existing datasets fall short on providing any information about specific traffic characteristics suitable for model probing, ... and the generation process for synthetic traffic usually does not provide sufficient structural nuances to explore the behaviour of models [18].

Scientific machine learning model development requires both **model evaluation**, in which the overall predictive quality of a model is assessed to identify the best model, as well as model validation, in which the behaviour and limitations of a model is assessed through targeted **model probing**. Model validation is essential to understand how particular data structures are processed, and enables researchers to develop their models accordingly. Data generation tools for rapid model probing in other domains such as the *What-If tool* [24] 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)*, tone and emotions can alter the meaning of a sentence significantly. The huge

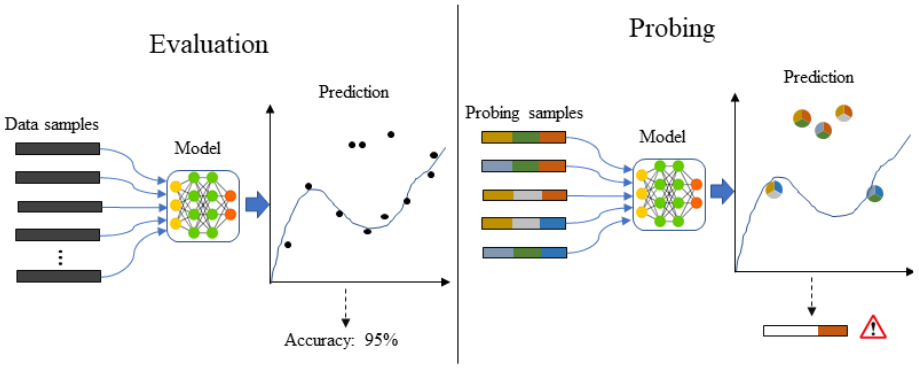


Fig. 1: Model evaluation and model probing with controlled data characteristics.

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) [14] 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 improved understanding between data and cyber-detection systems. Researchers today use threat reports to enhance behavioural malware labelling [20], 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.

This work provides the following contributions:

1. We present an exemplary dataset that is suitable for broad probing of models trained on the CICIDS-17 dataset citation here or later?, as it mirrors its range of protocols and attacks.
2. We demonstrate the probing of an LSTM-based intrusion detection model with this dataset, and demonstrate how to lower false-positives effectively by understanding where the model fails to process particular traffic structures correctly.

3. We examine how well DetGen is able to control different types of traffic characteristics, and compare the corresponding experimental determinism 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.

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

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 provides an overview over the probing dataset that we are releasing and explains why and in what way it mirrors the CICIDS-17 dataset. 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, before Section 6 concludes our work.

2 Motivation and Methodology

2.1 Scope of DetGen

Assume the following problem: You are designing a packet-level traffic classifier which is generating a significant amount of misclassification. These 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 the misclassified traffic events with these characteristics, which prevents you from identifying the misclassification cause by numeric evaluation. We designed DetGen to generate traffic with sufficient ground-truth information to allow effective association of such model failures with traffic characteristics.

In particular, we look at a *Long-short-term memory* (LSTM) network [1] by Hwang et al. [9], which is designed to classify attacks in web traffic. Through probing we will see, retransmission sequences in a packet sequence deplete the models classification accuracy. We

– We generate suitable probing traffic and input it to the trained model

[1] a deep learning design for sequential data

- We perform examine the correlation between traffic misclassification scores and the generated traffic microstructure labels, which is strongest for simulated packet loss.
- 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.

After training the model on the CICIDS-17 dataset [18], the biggest source for misclassifications turn out to be packet retransmission sequences.

Traffic is generated with precise control and monitoring over the conducted communication activity as well as various traffic-shaping factors. The scope lies on microscopic traffic-structures that become apparent on a packet- 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 4.2, 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.

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

In this use-case, we improve the design of a traffic classification model through the analysis of data separation in dependence of different traffic features. We use a recent traffic classification model by Hwang et al. [9] as an example, which aims at distinguishing various types of malicious activity from benign traffic. The model achieved some of the highest detection rates of packet-based classifiers in a recent survey [22] and is claimed to achieve detection and false-positive (FP) rates of **99.7%** and **0.03%** respectively. It classifies connections on a packet-level using a *Long-short-term memory* (LSTM) network [2].

In order to detect SQL-injections, we train the model on the CICIDS-17 dataset [18] (85% of connections), which contains 5 days of network traffic collected from 12 computers with attacks that were conducted in a laboratory setting. This dataset is the most recent and covers the most attacks of the several existing de-facto benchmark datasets. We also include a set of different HTTP-activities generated by DetGen (7.5%) that mirror the characteristics in the training, as explained in Section 3. Rather than providing an accurate and realistic detection setting, this example shows how traffic information can be linked to model failures and slumping performance. In total, we use 50,000 connections for training the model, or slightly less than 2 million packets.

[2] a deep learning design for sequential data

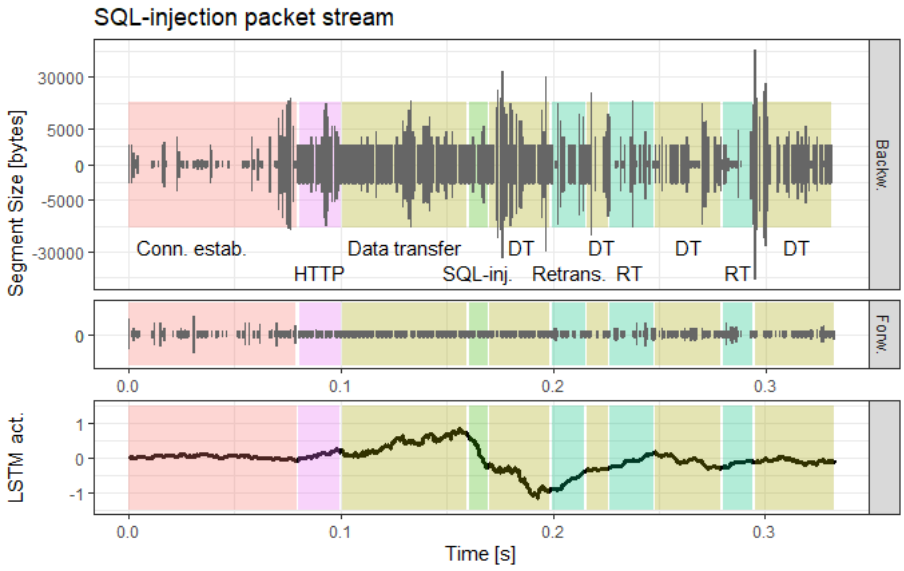


Fig. 2: LSTM-output activation in dependence of connection phases. Depicted are packet segment streams and their respective sizes in the forward and backward direction, with different phases in the connection coloured and labelled. Below is the LSTM-ouput activation while processing the packet streams.

The initially trained model performs relatively well, with an *Area under curve* (AUC)-score[3] of **0.981**, or a detection and false positive rate[4] of **0.96%** and **2.7%**. However, these rates are still far from enabling operational deployment.

Now suppose we want to improve these rates to both detect more SQL-injections and retain a lower false-positive rate. To start, we initially explore which type of connections are misclassified most often. We perform a correlation analysis between the numeric or categoric labels available for the probing data, and the binary response whether the corresponding connection was misclassified. Unsurprisingly, the highest correlation to misclassification was measured for the conducted activity, with a particular attack scenario (19% correlation) and connections with multiple GET-requests (11% correlation) being confused most often. This was followed by the amount of simulated latency (12% correlation), which we are now examining closer.

Fig. 3 depicts classification scores of connections in the probing data in dependence of the emulated network latency. The left panel depicts the scores for the initially trained model, which shows that while classification scores are well separated for lower congestion, increased latency in a connection leads to a narrowing of the classification scores, especially for SQL-injection traffic. Since there

[3] a measure describing the overall class separation of the model
[4] tuned for the geometric mean

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.

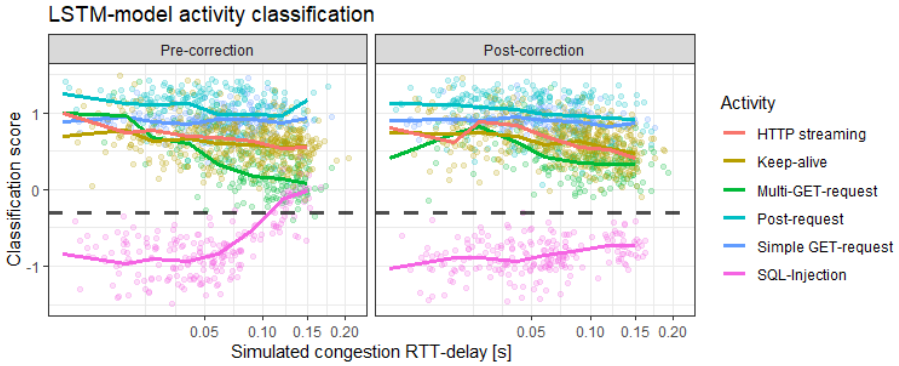


Fig. 3: Scores for the LSTM-traffic classification model in dependence of simulated network congestion, along with the classification threshold.

To examine the exact effect of retransmission sequences on the model output, we generate two similar connections, where one connection is subject to moderate packet loss and reordering while the other is not. We then compare how the LSTM-output activation is affected by retransmission sequences. Fig. 2 depicts the evolution of the LSTM-output layer activation in dependence of difference connection phases. Initially the model begins to view the connection as benign when processing regular traffic, until the SQL-injection is performed. The model then quickly adjusts and provides a malicious classification after processing the injection phase and the subsequent data transfer. The negative output activation is however quickly depleted once the model processes a retransmission phase, and is afterwards not able to relate the still ongoing data transfer to the injection phase. When comparing this to the connection without retransmissions, we do not encounter this depletion effect, instead the negative activation persists after the injection phase.

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

0.997 while tuned detection rates and false positives improved to **99.1%** and **0.045%**.

3 Probing dataset

Do we need a name for this dataset?Where to post link?

3.1 Mirroring CICDS-17 dataset

We compiled and released a dataset suitable to quickly probe ML-model behaviour on a range of traffic characteristics. This dataset is designed to contain similar traffic as the CICIDS-17 [18] dataset to allow probing of models trained on this dataset in a similar manner as demonstrated in Section 2.1. For this, we mirrored several high-level properties to provide the same traffic structures the models learned. In particular, we mirrored the following properties:

- **Application layer protocols**: We used the same range of major protocols that occur in the CICIDS-dataset, namely *HTTP/SSL, SMTP, FTP, SSH, SQL, SMB, and NTP*. We excluded some protocols such as DNS, LDAP, or Kerberos since these do not occur in sufficient amount and complexity in the CICIDS-17 dataset. Table in Appendix, to be added displays the corresponding frequency of protocols.
- **Implementations**: For each of the protocols, we used similar application implementations, such as *Apache* for HTTP-activities, *VSFTP* for FTP-activity, or *OpenSSH* for SSH.
- **Conducted attack types**: We aimed to cover as many attack types included in the CICIDS dataset as possible. These include *SQL-injections, SSH-brute-force, XSS, Botnet, Heartbleed, GoldenEye, and SlowLoris*. We were not able to cover all attacks though as DetGen either did not provide the necessary network topology to conduct the attack, such as for port-scanning, or the attack types are not implemented in the catalogue of scenarios yet.

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

In Fig. 4 we compare the validation error of a recent neural network model for network intrusion detection [3] on the probing dataset. We distinguish models when trained exclusively on the CICIDS data, and when also trained on the probing data. Even though the validation error is slightly higher when only trained on the CICIDS data, the difference is almost negligible compared to the error resulting from a model trained on a completely different dataset (UGR-16 [15]). This does not fully proof that every model is able to transfer observed structures between the two datasets, but it gives an indicator that they mirror characteristics.

3.2 Dataset statistics

To be added

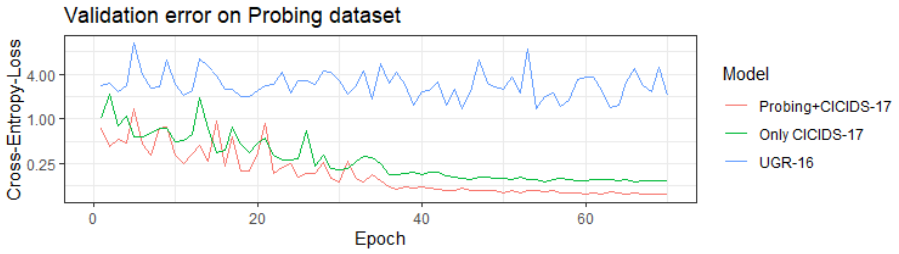


Fig. 4: Validation errors of LSTM-model [3] on Probing dataset when trained exclusively on the CICIDS-17 dataset (green) and when also trained on the Probing dataset (red), as well as when trained on a third unrelated dataset for comparison (blue).

4 DetGen and traffic microstructures

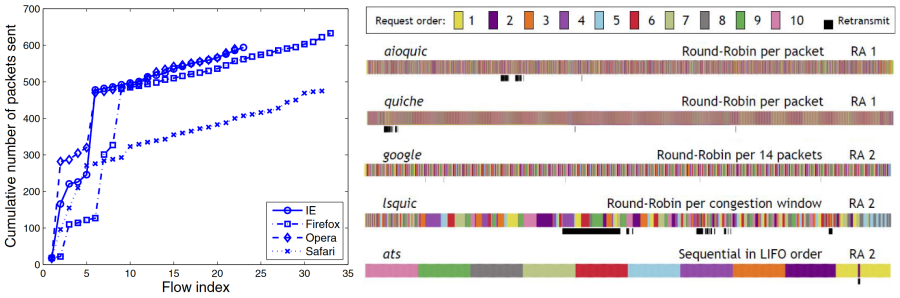
4.1 Traffic microstructures and their influence factors

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. We highlight and quantify the most dominant ones here, which will act as a justification for the design choices we outline in Section 5.4.

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. [21, 25]. Fig. 5a provides as an example the packet-per-connection differences between different browser choices.

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. [16] for the QUIC/HTTP3 protocol. Fig. 5b depicts differences in channel prioritisation for different QUIC-implementations.

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.



(a) Number of packets sent from a browser, accumulated over all flows that comprise the retrieval. [16].
(b) Comparison of QUIC connection request multiplexing for selected implementations, taken from [25].

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. 6, where the host sends significantly less `ack`-packets when under heavy computational load.

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].

We designed DetGen to control and monitor these factors in order to let researchers explore their impact on their traffic models. We omitted some factors that can influence traffic structures, since these act either on a larger scale or correspond to exotic settings that are outside of our traffic generation scope. Among them are the following:

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

2. Networking stack load TCP or IP queue filling of the kernel networking stack can increase packet waiting times and therefore IATs of the traffic trace, as shown by [17]. In practice, this effect seems to be constrained to large WAN-servers and routers, and we did not find any effect on packet-sequences for various amounts of load generated with iPerf for a regular UNIX host.

3. Network configurations Network settings such as the MTU or the enabling of TCP Segment Reassembly Offloading have effects on the captured packet sizes,

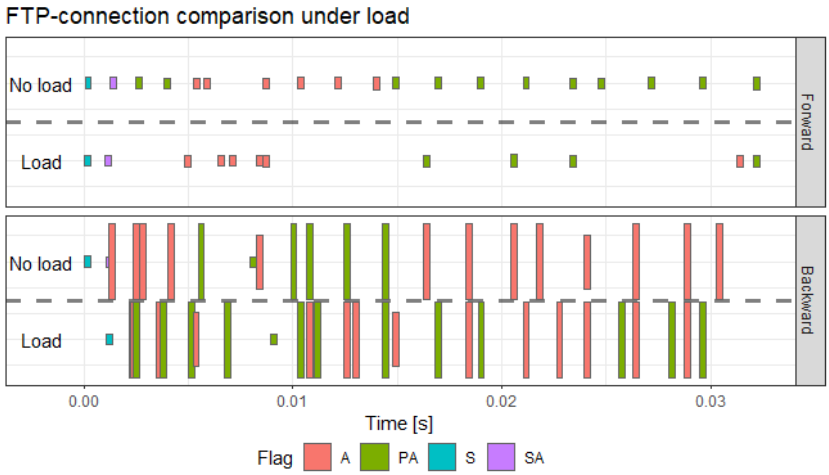


Fig. 6: Packet-sequence structure similarity comparison for FTP-activity under different load and otherwise constant settings. Colours indicate packet flags while the height of the packets indicates their size. Note that under load, the host sends significantly less packets.

but are standardised for most networks, as documented in the CAIDA traffic traces [23].

4.2 Traffic control and generation determinism

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. 9. 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:

– **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 [12], and measure the cosine similarity between connections, which is also used in general traffic classification [1].

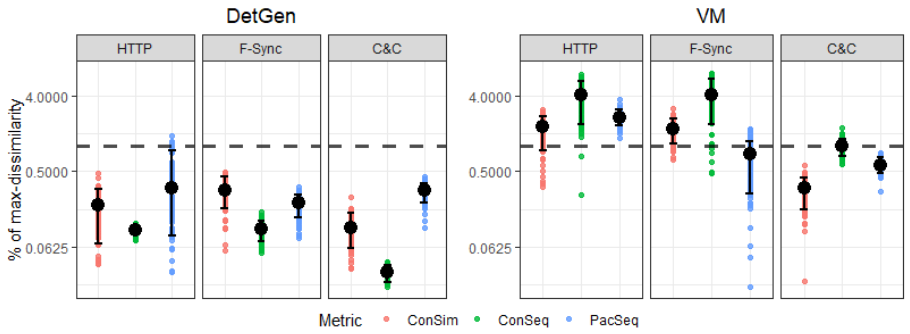


Fig. 7: Comparison of HTTP-group dissimilarity scores for the DetGen-framework and a regular VM-setup, on a logarithmic scale. Samples from the VM-setting are consistently more dissimilar, in particular for flow-based metrics, where the average dissimilarity is more than 30 times higher than for the DetGen setting.

- **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. [25] 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 scores yield less than 1% of the dissimilarity observed on average for each protocol. Scores are especially low when compared to traffic groups collected in the VM setting: Dissimilarity scores for the VM-setting are most notably higher for the connection-metric, 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. 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.

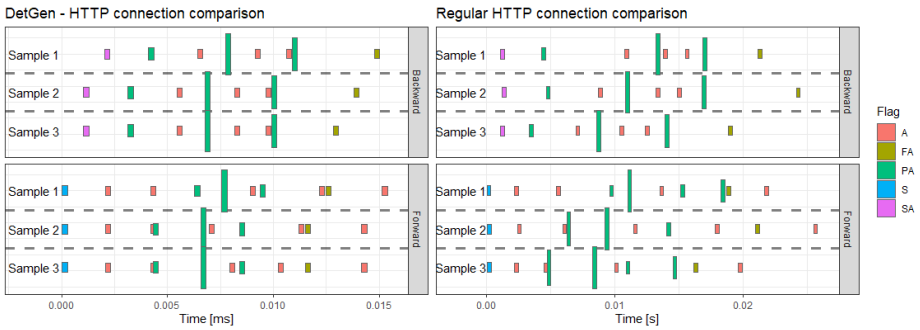


Fig. 8: Packet-sequence structure similarity comparison for HTTP-activity under constant settings generated by the DetGen framework (left) and in a regular setting (right). Colours indicate packet flags while the height of the packets indicates their size. Note that in addition to more differences in the timing, the packet sizes vary more in the regular setting.

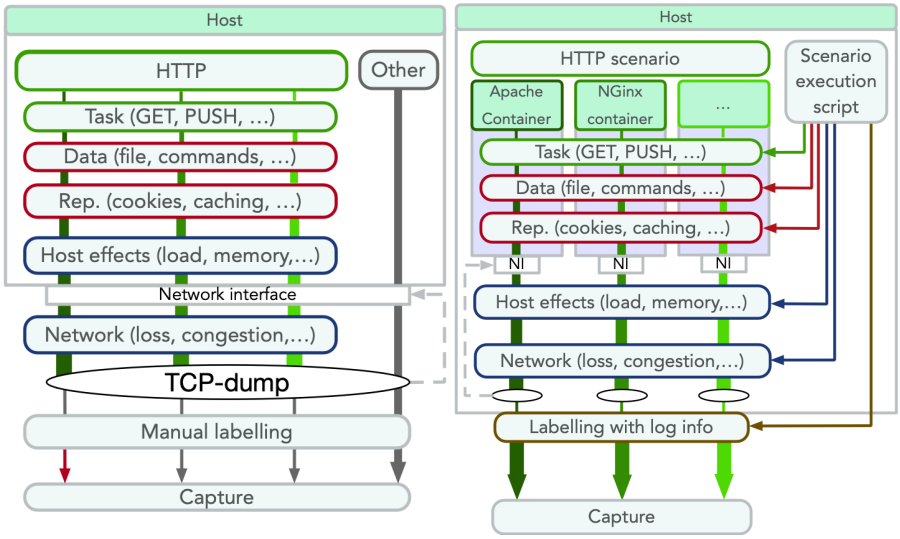
5 DetGen Architecture

5.1 Design overview

Detgen is a container-based network traffic generation framework that 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. 9 provides a technical design comparison of the DetGen-setup and traditional VM-based setups and highlights how control and monitoring is exerted.

5.2 Containerization and activity isolation

As we demonstrated in Section 4.2, 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



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

Fig. 9: Design comparison of traditional NIDS traffic-generation-setups (a), and our DetGen framework (b).

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. [19]. 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 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 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.

6.1 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 [15] use this approach to fuse real-world and synthetic traffic and are currently better suited to build models of large-scale traffic structures.

Furthermore, while controlling traffic shaping factors artificially helps at identifying the limits and weak points of a model, it can exaggerate some characteristics in unrealistic ways and thus both affect the training phase of a model as well as tilt the actual detection performance of a model in either direction. Additionally, the artificial randomisation of traffic shaping factors can currently not generate the traffic diversity encountered in real-life traffic and thus only aid

at exploring model limits extensively. The lack of realistic traffic heterogeneity however is at the moment significantly more pronounced in commonly used network intrusion datasets such as the CICIDS-17 dataset, where the vast majority of successful FTP-transfers consist of a client downloading a single text file that contains the Wikipedia page for 'Encryption'.

6.2 Future work

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 [13] 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.

6.3 Acknowledgement

We are grateful for our ongoing collaboration with our industry partners on this topic area, who provided both ongoing support and guidance to this work. Discussions with them have helped reinforce the need for a better evaluation and understanding of the possibilities that new intelligent tools can provide.

Full funding sources after currently blinded.

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. 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.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.

21. 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.
22. 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.
23. C. Walsworth, E. Aben, K. Claffy, and D. Andersen. The caida ucsd anonymized internet traces 2018,", 2018.
24. 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.
25. 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% (CIC-IDS 2017), 65.6% (UNSW NB15), and 94.5% (ISCX Botnet) of network flows in the respective dataset. Our evaluation shows that some protocols that make up a substantial amount of real-world traffic are glaringly omitted by current synthetic datasets, such as BitTorrent or video streaming protocols, which we decided to include.

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

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

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

B CICIDS-17 statistics

Name	Description	#Ssc.
Ping	Client pinging DNS server	1
Nginx	Client accessing Nginx server	2
Apache	Client accessing Apache server	2
SSH	Client communicating with SSHD server	5
VSFTPD	Client communicating with VSFTPD server	12
Wordpress	Client accessing Wordpress site	5
Syncthing	Clients synchronize files via Syncthing	7
mailx	Mailx instance sending emails over SMTP	5
IRC	Clients communicate via IRCd	4
BitTorrent	Download and seed torrents	3
SQL	Apache with MySQL	4
NTP	NTP client	2
Mopidy	Music Streaming	5
RTMP	Video Streaming Server	3
WAN Wget	Download websites	5
SSH B.force	Bruteforcing a password over SSH	3
URL Fuzz	Bruteforcing URL	1
Basic B.force	Bruteforcing Basic Authentication	2
Goldeneye	DoS attack on Web Server	1
Slowhttptest	DoS attack on Web Server	4
Mirai	Mirai botnet DDoS	3
Heartbleed	Heartbleed exploit	1
Ares	Backdoored Server	3
Cryptojacking	Cryptomining malware	1
XXE	External XML Entity	3
SQLi	SQL injection attack	2
Stepstone	Relayed traffic using SSH-tunnels	2

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