Università degli Studi di Pisa

GRADUATION THESIS

Detecting network anomalies using the feature space latent representation

---

Thesis Submitted for the *Master Degree in Computer Science*,

Artificial Intelligence Curriculum

Department of Computer Science

Supervisors:

Luca Deri, Giuseppe Attardi

Author:

Samuele Sabella

Academic Year 2019/2020

Contents

1 Introduction

1.1 Network anomaly detection

Intrusion detection systems (*IDSs*) are popular topics in research with many surveys stating the problem and overviewing prior work [Chalapathy et al., 2019][Moustafa et al., 2019]. The main goal can be summarized as to discriminate between normal user activity and malicious or unusual one. What is to be considered normal is, of course, still a debate.

IDSs can be classified into two main families: *misuse-based (M-IDS)* and *anomaly based (A-IDS)* [Sommer et al., 2010]. The former assumes a known set of malicious behaviour and examine the data available to look for signatures suggesting an attempted intrusion or a successful attack. Due to encryption, creating signatures is becoming harder and more expensive, constraining IDSs to decrypt network traffic or to look for different metrics other than packet payloads [Yaacoubi, 2019]. Moreover M-IDSs flaw to detect previously unknown threats. To overcome this lack A-IDSs aim to build up a model of normal system behavior and report any deviation from this known good pattern. Typically, A-IDSs monitor a device for a reasonable amount of time and then use the collected data to learn how the device behaves under normal conditions. A *supervised* A-IDS uses data labeled by humans or in an automatic way to learn how to discriminate between good and malicious behavior, whereas an *unsupervised* approach uses the data collected during the monitoring phase without any additional information concerning what is normal and what is to be considered an anomaly. Even if A-IDSs can detect previously unseen anomalies, current A-IDSs are prone to false positives and often require non-negligible resources such as time for learning and ad-hoc hardware to achieve reasonable performance. Recent studies in the field of *adversarial attacks* have also raised an additional concern due to the fact that this type of systems may be fooled by an attacker [Lin et al., 2018]. Thus, M-IDSs remain widespread. IDSs can also be subdivided by examining the data used. *Host IDSs (HIDSs)* use data such as logs, system calls traces or any other data collected on the monitored machine, whereas a *Network IDS (NIDS)* use only the data collected by monitoring network traf-

fic.

Most of the work in the literature concerning anomaly based intrusion detection systems addresses supervised anomaly detection. However, this approach has the major shortcoming of creating models biased towards the dataset, which is often too small, doesn't reflect the data collected in real-case scenarios and may become quickly deprecated due to fast changes in the current technologies [Hindy et al., 2020]. Indeed, labelling data is often costly or impossible due the volume of the data itself. Moreover employing a probe on the host being monitored to collect information beyond network traffic is also unfeasible in many situations. To overcome these issues this work aims to study the unsupervised *anomaly based network IDS (A-NIDS)* problem in which we are provided only with raw traffic data collected from many devices and no clues on how the anomalous traffic may look like.

Most of the techniques used in unsupervised A-NIDS does not take into account the temporal nature of network monitoring or use models, like holt-winter [Winters, 1960], which are unable to correlate many network metrics at once and cannot combine the data collected from different devices to build up a general knowledge of how devices usually behaves. Autoencoders are the main artificial neural network architecture used for unsupervised A-NIDS and address this issues, however they can be expensive in terms of resource usage. Moreover, most of the tools available does not offer additional control beyond computing an *anomaly score* that measures the probability of a sample being an outlier. We propose a new learning framework based on multivariate time series representation learning for network activity. The latent representations computed by our model can be used for: a) create a fingerprint of the device behavior at a given time; b) detect anomalous behaviors, i.e. those events that are not coherent with the behavior expected by a device or are very close to known bad ones; c) finding similarities between devices and their behavior.

To summarize, our main contributions are:

- An unsupervised anomaly detection tool able to combine the information collected from different devices and to correlate different network metrics.
- A comparative study of different techniques available to address unsupervised

A-NIDS.

- The proposal of a methodology to provide network administrators additional tools based on device behavior similarity which enable to look for similar events in different networks or in the historical data.

The rest of the document is organized as follows. Recent works in unsupervised *A-NIDS* is described in subsection 1.2. Our methodology is described in section 2 and experiments are reported in section 3. Finally, conclusions are drawn in section 6.

1.2 Related work

The limits that supervised methods currently have made unsupervised *A-NIDS* a field growing in popularity. In [Usama et al., 2019] a comprehensive view of the problems and current solutions is given. The authors of [Dromard et al., 2019] compared outliers detection algorithms using a subsample of the famous *KDD99* [Hettich, 1999], a dataset containing intrusions simulated in a network environment. From their study they pointed out that using *robust principal component analysis (PCA)* [Kwitt et al., 2007] to detect anomalous connections achieve great detection accuracy with significant parameter robustness and high performance. Robust PCA is able to reduce the impact of outliers in the training set, thus leading to better performance compared to classical PCA. However, it is worth to say that in the very same study they pointed out that anomalous samples in the dataset had an extremely skewed value in at least one dimension of the feature space. Thus, even a naive algorithm can achieve astonishing accuracy on KDD99. Further analysis on a different dataset is left as future work. The authors of [Dromard et al., 2016] proposed *ORUNADA*, a near real-time evolution of *UNADA* [Casas et al., 2011]. As its predecessor, ORUNADA aggregates network packets into flows using a key such as source or destination IP. Flows are described by a set of d statistics, or features, which are then partitioned into $\binom{d}{2}$ subgroups. Density based clustering algorithm is then applied to each pair of features. To achieve reasonable speed they used a sliding window and applied incremental grid-based clustering (*IDGCA*) [Chen et al., 2002] over the feature space at each sliding window step.

Dense regions in the feature space are grouped together and a threshold is used to remove small clusters. When IDGCA has been applied to the whole sliding window, the detection algorithm is started. The anomaly score of flow x is computed as the sum of the distances $d(x_s, C)$, with s a subspace in which x is considered an outlier, x_s the projection of x in that subspace, C the biggest centroid in s and d a distance measure. We point out that computing every possible combination of the features extracted by each flow may result in an excessive waste of resources due to the fact that many features are poorly correlated. Furthermore using binary groups of features may leave out interesting correlations which involves more than two features.

A semi-supervised technique is proposed by the authors of [Aygun et al., 2017]. Artificial neural network autoencoders (*AE*) [Rogers and McClelland, 2014] and denoising autoencoders (*DAE*) [Vincent et al., 2010] have been compared to classify a flow as anomalous using the reconstruction error computed over the features provided by the dataset used, NSL-KDD [CSE, 2009]. The algorithm is trained on normal traffic samples, then a threshold is optimally chosen using a labeled validation set composed of normal and anomalous traffic. Even if the model is trained in a semi-supervised environment, the results shows that the autoencoder architecture can achieve results comparable to supervised techniques. [Kathareios et al., 2017] developed an encryption aware *A-NIDS* which aims to reduce the number of samples that have to be examined by an expert. Packets sent by a common source, i.e. flows, are grouped into fixed and non-overlapping time intervals. A multivariate time series is collected by computing different statistics on each time interval. Only the information available at *layer-2* to *layer-4* in packets' header and metrics computed using network flows themselves (e.g., packets per second rate) is used. Anomaly detection is performed in three steps: *a)* an autoencoder is trained to reconstruct each point in the multivariate time series; *b)* points with high reconstruction error are collected in a set of potential anomalies, called A, and a sub-sample of that set is classified by an expert. *c)* The neighbors of each anomaly point in A are searched among the samples labeled by the expert. *d)* If the number of threat in the neighborhood region of an anomaly point is over a threshold C, then the point is classified as an anomaly. Good results are achieved but a labeled

validation set is still needed to tune the anomaly threshold C. The authors of [Mirsky et al., 2018] proposed a lightweight distributed *NIDS*, called *Kitsune*, which aims to operate on devices subject to performance constraints like routers. A multivariate time series with 110 channels, each representing a network metric, is computed online and a correlation matrix is kept updated while the traffic flows through the network. The correlation matrix is used to identify small subsets, each of size K, of highly correlated channels from the entire features set. A first set of simple three-layer autoencoders is trained to take as input a subset of features at time step t and to minimize the reconstruction error. Then, another three layer autoencoder is used to reconstruct the *root mean square error* (*RMSE*) computed for each autoencoder at the previous step. The autoencoder in the second layer is used to enhance the missed correlations among different sets of features instead of using the sum of individual reconstruction errors. Using multiple autoencoders, each one reconstructing a set of highly correlating features, helped the authors to keep the complexity low and the system able to process a maximum of 5400 packets per second on a low-powered devices (i.e. Raspberry Pi). Even if Kitsune achieves good results without any labeled data, we suggest that taking into account the temporal nature of network traffic data can undercover malicious patterns that can't be discovered using a point-wise anomaly detection technique. To overcome this problem, [Mirza et al., 2018] used an *LSTM* [Hochreiter and Schmidhuber, 1997] based encoder-decoder architecture to reconstruct the hexadecimal values in the packets' payload. Once again, a threshold over the reconstruction error is used to detect and report anomalies. Even if [Mirza et al., 2018] take into account the sequential nature of network data and traffic monitoring, they don't take into account encrypted traffic which nowadays makes up most of the network traffic. The authors of [Ergen et al., 2019] tried to improve the mechanism for selecting the threshold above which model errors are considered anomalies. Driven by the fact that using artificial neural models to detect anomalies by learning how a time series behaves and output an anomaly score as a function of the prediction error leads to the use of probabilistic models and a user-defined threshold [Malhotra et al., 2015], they experimented the use of LSTMs and *GRUs* [Cho et al., 2014] together with *SVDD* [Tax and Duin, 2004]

and *OC-SVM* [Schölkopf et al., 2001]. The hidden states produced by the LSTM are encoded in fixed length vector using mean-pooling, then they are used as input for an SVDD or OC-SVM model. The system is trained end-to-end with SGD or quadratic programming-based approaches. Their approach seems very promising, however they lack a large scale test on realistic network data. Another technique commonly used to solve network anomaly detection involves the use of auto regressive models, such as an auto-regressive integrated moving average (*ARIMA*) [Zhou et al., 2005], or models that accounts for seasonality, such as Holt-Winters [Pena et al., 2013]. Typically, these techniques allow to define an upper and lower uncertainty interval on the model predictions which can be used to detect anomalies whenever samples do not fall in the range predicted by the model. Besides the high speed performance that can be reached using such techniques, they are unable to generalize the knowledge acquired to previously unseen hosts. To make an example, if we aim to detect anomalies generated by a host, we need clean network traffic collected previously for that host. Having clean network traffic for an infected or strangely behaving host is unfeasible in many scenarios and, as shown in our experiments, can be avoided using machine learning based methods.
We propose a time and encryption aware unsupervised model based on triplet loss minimization, with minor modifications to the one proposed by [Schroff et al., 2015]. The use of such objective function enables us to avoid a threshold tuning phase over labeled data as done previously for other techniques such as autoencoders. We tested our model on the data provided in the CIC-IDS2017 dataset [CIC, 2017] and compared our results to existing methods: sequence to sequence autoencoder, *Kitsune* and *Prophet*, the newest seasonality based forecasting model promoted by Facebook [Taylor and Letham, 2018]. We describe implementation details of each of them in subsection 4.1, 4.2, 4.3. For both triple loss and autoencoders we took advantage of the latent representation given by the model to visualize the behavior of each device over time and to cluster similar events, thus resulting in new ways of monitoring network traffic.

1.3 Motivation

Almost all anomaly detection systems in literature and for commercial use do not integrate knowledge of network activity between different devices. They are mainly based on mathematical models, such as ARIMA, which are fitted on the device at hand and don't extract general knowledge about what has to be considered normal network activity behavior. This work aims to study a new setting in which the model has to learn how a group of devices $\mathscr{D}$ behaves by integrating the knowledge extracted from each device. The knowledge acquired is used to detect anomalies generated by a previously unseen device. We have explored ways of achieving this goal using unsupervised techniques which we believe are the best choice in the field of network monitoring where labeled data is, at the present time, scarce. Moreover, we think that the temporal nature of network activity is crucial to detect some scenarios and underestimated by the current technology [Kwitt et al., 2007], thus we build our solution using time-aware models. In the conclusion of this paper (section 6) we detail how this new setting can lead to overcome the current difficulties posed by the scarcity of the data and often unrealistic scenarios found in the modern datasets.

To integrate the knowledge about different devices our work aims to create digital fingerprints of network activities which unambiguously identifies devices and their behavior over time. Fingerprints are aggregated into clusters of similar events and anomalies are detected as drastic changes between near in time fingerprints. We argue that the next step towards better network monitoring is to further group information as done previously in history. Packets have been grouped into network flows to prevent administrators from being overwhelmed by information which, in turns, leads to bad network monitoring. We aim to group time series describing network behavior into a fingerprint of the host being monitored in such a way that it can be visualized and compared to other hosts. We argue that an alert should be raised whenever a change in the behavior signature is discovered while monitoring a device. Network administrators should then proceed in further investigation using the signature which identifies the anomalous behavior to find similar activity in other parts of the network or in the historical data. More granular information, like flows or packets, can then be used to

investigate the causes.

In our work we are not addressing effective attacks pattern recognition, which can't be generalized to unknown attacks. We are addressing anomaly detection which include attacks but also misconfigurations, people habit changes and all other anomalies which can be detected by looking only at time series generated by network monitoring tools. We believe machine learning is able to achieve this goal.

2 Methods

2.1 Solution architecture

We define an anomaly according to [Ergen et al., 2019], i.e. *an unexpected event that cannot be predicted from the data observed in the recent past*. Our study concerns network monitoring, thus, we focus either in detecting attacks (floods, DDoS incoming or outgoing), misconfigurations (firewall, DNS) or changes in user habits both in timetables or in network usage (DNS replaced with DoH).

Given a device, we monitor its network activity by means of a set of metrics collected over time using *ntopng* [ntop, 2017], a popular flow-based network monitoring tool. Bytes sent and received or the number of active flows (figure 1) are good examples of the metrics collected. Further details are given in subsection 3.1. We have chosen ntopng because it is able to provide us with a rich multivariate time series that can deeply describe the behavior of a host. Furthermore, ntopng is able to process multi-gigabit connections which are often used in public and private environments, thus it does not set us constraints on the kind of environment we aim to target.

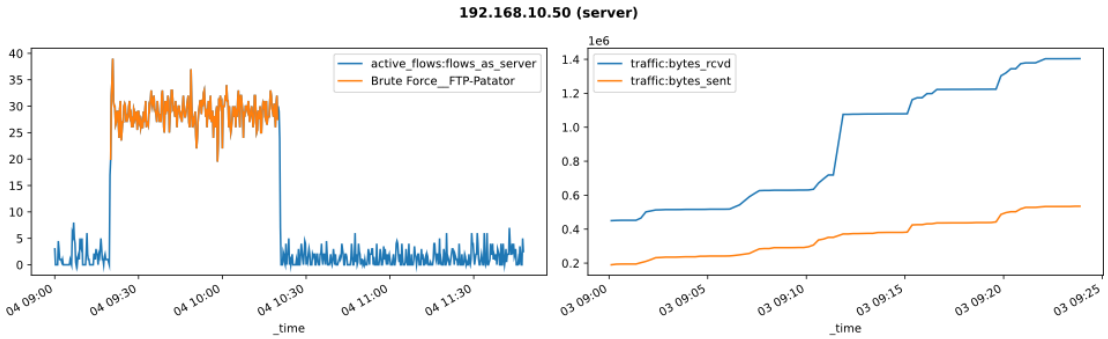

Figure 1: **Left:** number of active flows as server during a bruteforce attack; **Right:** bytes sent and reveived during normal activity

We are addressing the problem of multivariate time series anomaly detection, where each time series contains network statistics computed for a single host. Given a multivariate time series, we solve the problem by creating a fingerprint of the device behaviour for successive time windows and then by reporting an anomaly if the fingerprints computed are significantly different than expected. That is, our main objective isn't to learn how to discriminate between known good and bad traffic, which wouldn't generalize to previously unknown attack vectors different from the one seen in training [1]. Instead, we aim to create a model which, given an unknown device, will return us a latent representation which fully describes how the device is behaving right now and how it may evolve its behavior in the recent future. If the future behavior doesn't match the expectation, we raise an anomaly alert. Thus, we want to introduce the concept of network behavior similarity and, eventually, allow administrators to find all the devices that exhibit a common behavior, which may eventually be flagged as anomalous or not. In our settings, at training time the system is provided with a dataset $\mathscr{D}_{tr} = \{S_1, ..., S_N\}$ containing N multivariate time series, each generated by monitoring the network activity of one host for a continuous period of time. It is important to note that $\mathscr{D}_{tr}$ contains the time series generated by multiple hosts and even the same host but at different times as shown in figure 2. What matters is that each time series has been generated by monitoring a single host for a continuous amount of time.

[1] A system trained to discriminate between incoming DoS and normal traffic will struggle to detect outgoing DoS made by the device being monitored

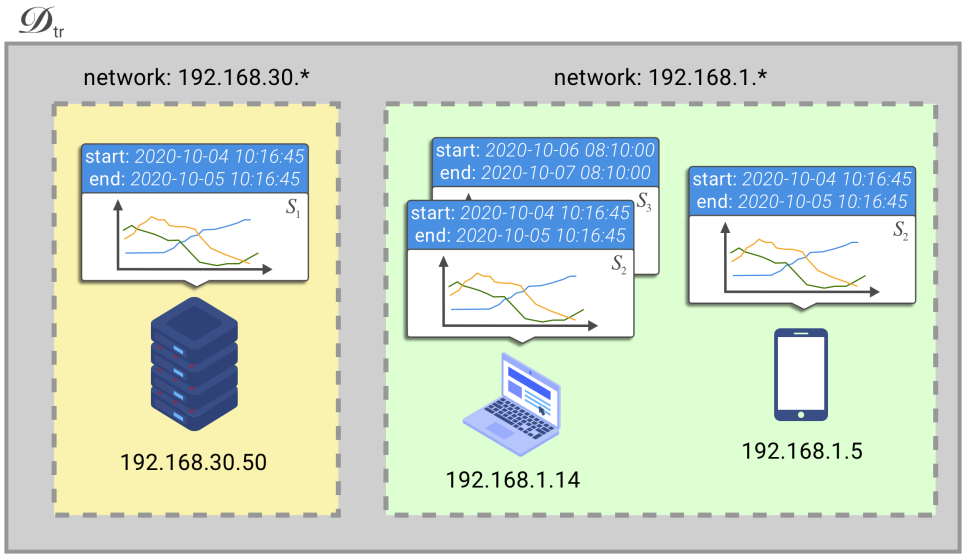

Figure 2: The dataset contains time series of network activity features computed with *ntopng* for multiple devices, eventually from different networks.

Each multivariate time series, $S_d =< X_1, ..., X_{k_d} >$, contains k_d samples, each with c channels representing the network metrics extracted by *ntopng*. An artificial neural model is trained to learn the normal behavior of devices by trying to cluster time series samples in a latent space. To this end we used a modified triplet loss whose discussion is postponed to section 2.2. When the model has learnt how to discriminate the time series of different devices in the latent space, it is used to create the latent representation of previously unseen devices. The latter are monitored by computing the latent representation of a rolling window of the multivariate time series generated by *ntopng*. If the latent representation doesn't evolve as expected an alert is generated and the most similar events are returned together with an anomaly score generated by comparing fingerprints from the recent past with the last fingerprint computed. Figure 3 shows how our architecture is meant to be used at deploy time.

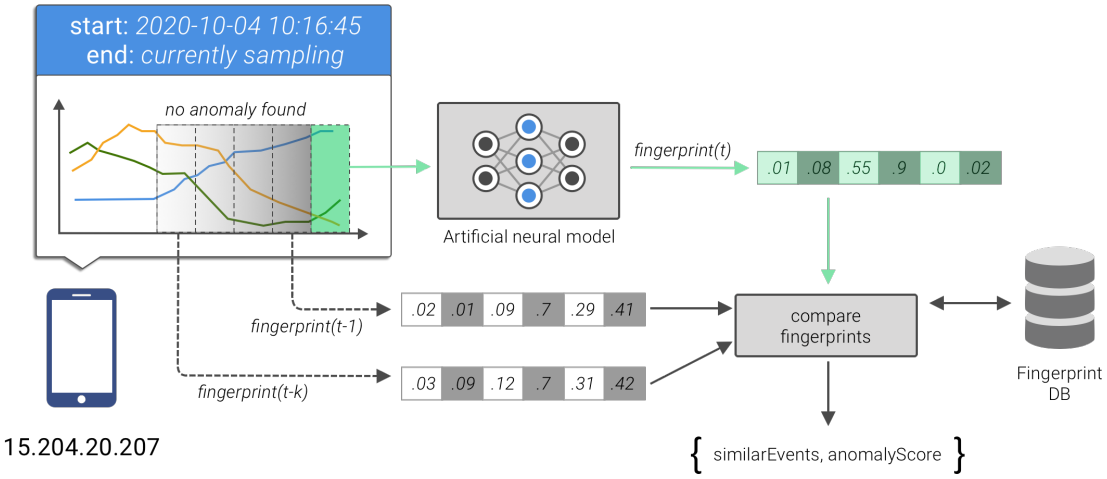

Figure 3: When deployed, the model encodes time series samples into a latent representation (fingerprint). The system uses the fingerprint to return an anomaly score together with similar events

We aim to create a model able to combine the information collected from different hosts into a latent representation containing general knowledge about usual network activity. Once this knowledge is acquired it can be used to detect if a device is not behaving as expected. Thus, our model is not meant to be limited only to the extent of the network or device being monitored as usually happens in commercial A-NIDS. Because of our learning framework, methods like ARIMA and Prophet can't be used at all. Indeed, using such models doesn't allow us to combine effectively information coming from different sources (i.e. devices), moreover we may completely lack normal traffic for the device outside of the training set. Thus, applying Prophet to the unknown devices' traffic would result in a poor anomaly detection score because the traffic we would train on may contain both normal activity and attacks.

2.2 Clustering in the latent space

At the end of the data extraction and preprocessing step we are provided with a multivariate time series for each host in the network similar to the ones shown in figure 1 (right). Inspired by the work of [Franceschi et al., 2019] which applied triplet loss to time series data, and [Banville et al., 2019] which takes advantage of time series ordering and time distance to introduce a self supervised loss function, we looked for

unsupervised techniques to address network anomaly detection. From now on we will refer to a 20 minutes multivariate time-series sample of a target host as *context window* and an *activity* as a 10 minute sub-sample of it. We designed our training objective, based on the following intuitions: *a)* A host won't change radically its behavior in a short time. Indeed, we expect that each host shows a peculiar but standard behavior over time. If all time series were random, we would not be able to define what is normal for a given host. In other words: *"without a normal behavior we cannot detect anomalies". b)* Learning normality is different from learning to discriminate between known good and known bad traffic. The aim of learning the concept of normal with as broad a connotation as possible requires an unsupervised or self-supervised technique. Thus, we may use frameworks like autoencoders or next-step-ahead predictors, however all this technique requires the model to learn every single detail in the time series in question. We argue that this fine-grained objective is a too difficult task in the framework of network monitoring and may results in additional, unnecessary, complexity. Therefore we propose that we should only know if what we sample at a given moment in time is coherent with respect to what has been learnt from the device behavior and what just happened in the near past. *c)* The traffic should be coherent: the latent representation of activities near in time should be similar. Furthermore, the latent representation of an activity should give information about its context. *d)* An anomaly is something that is not expected to happen. Thus, to detect anomalies we aim to check if two successive activities are coherent or not.

To address this intuitions we used a modified triplet loss. Throughout the following paragraphs we will mark in bold matrices representing multivariate time series. Given $\mathbf{c} \in \mathbb{R}^{T,m}$ a context window with m channels and T equal to 20 minutes, we randomly sampled a 10 minute activity $\mathbf{s} \in \mathbb{R}^{t,m}$ from $\mathbf{c}$. Then, given f_{emb} an encoder module, able to take as input a multivariate time series and give as output a fixed size representation of it, we defined a first loss term, L_{ctx}, to enforce the latent representation of $\mathbf{s}$ to be closer to the one of the surrounding context $\mathbf{c}$ than a margin α_1 by minimizing:

$$L_{ctx}(\mathbf{s},\mathbf{c}) = ReLu(||f_{emb}(\mathbf{s}) - f_{emb}(\mathbf{c})||_2^2 - \alpha_1) \tag{1}$$

To address intuition a and avoid the embedding space to collapse into one point we trained the model to project the latent representation of an activity coming from a different host, namely **an**, at a distance greater than a second margin α_2, thus resulting in a second loss term L_{coh}:

$$L_{coh}(\mathbf{s}, \mathbf{an}) = ReLu(||f_{emb}(\mathbf{s}) - f_{emb}(\mathbf{an})||_2^2 + \alpha_2) \qquad \text{with} \qquad host(\mathbf{s}) \neq host(\mathbf{an}) \tag{2}$$

Due to the fact that the probability of picking **an** at random and having the distance between the latent representation of **s** and **an** closer than the margin α_2 is very low, we picked **an** to be the closest example to **s** in the batch [Schroff et al., 2015]. This is typically referred to as *hard triplet sampling*.

The sum of L_{ctx} with L_{coh} results in a modified triplet loss function which in our case prevented the latent space to collapse:

$$L(\mathbf{s}, \mathbf{c}, \mathbf{an}) = L_{ctx}(\mathbf{s}, \mathbf{c}) + L_{coh}(\mathbf{s}, \mathbf{an}) \tag{3}$$

To detect anomalies we applied density based clustering in the latent space and assumed that most of the traffic is benign, thus classifying everything that does not belong to the largest cluster as anomalous. We suggest that the triplet loss leads to another, simpler and faster way to detect anomalies. Indeed, it should be noted that, by definition, successive activities are driven by the objective function to be at a distance less than two times α_1. It is straightforward to derive an anomaly score by normalizing the distance between two successive activities within the margin $[2 * \alpha_1, \alpha_2]$. This leads to an anomaly detection methodology which doesn't look for the single outlier point in a time series but to smooth qualitative changes within it. However this prevents us from comparing our framework with existing tools, thus we have discarded it.

3 Validation

3.1 Data extraction and preprocessing

Most of the models proposed in literature are trained with features given by the dataset used, which often provides basic network information such as the network flows length

and packets specific metrics. We employed *ntopng* to collect a rich multivariate time series for each host in the network. Using powerful tools such as ntopng has the major advantage of providing the model with high level features which go far beyond basic statistics computed on the packets flowing through the network. The number of flows that do not meet the expected TCP behaviour (*misbehaving_flows*), the number of times a host failed to communicate with a remote server/client (*host_unreachable_flows*) and the number of DNS queries failed (*dns_qry_sent_rsp_rcvd:replies_error_packets*) are good examples of what can be extracted using ntopng. Among the many, ntopng provides also *deep packet inspection (dpi)* features to collect L7 categorized traffic information for each host in the network. Thus, we may also have taken advantage of a multivariate time series containing data coming from L7 applications ranging from Skype, Whatsapp to Telnet. However, we have chosen to avoid the use of dpi data because the dataset we used in our experiments is entirely focused on detecting network attacks, thus this kind of information is superfluous. We think that a future work which takes into account people's habits changes will take advantage also of these very important ndpi-features that are worth mentioning. Ntopng provides also data coming from alerts generated by user-defined thresholds. We decided not to use user-dependent metrics because they may bias the model towards bad decisions and, even more important, prevent us from having an unbiased performance test due to the possible tweaking of user-defined thresholds to achieve better results.

Aimed by the fact that our experiments cover mostly network volumetric attacks (due to the dataset at hand, section 3.2), we have chosen a subset of all the features provided by ntopng to maximize our attack detection capabilities. The choice has been made by looking at which features are the most influenced while a network attack occurs. We have left the testing of the entire feature set as future work with the hope that a new, larger and more complex dataset will be realised in the future. Table 3 reports the feature set used in all our experiments.

metric	name	description
active_flows	flows_as_client	Number of active flows with the host as client
	flows_as_server	Number of active flows with the host as server
misbehaving_flows[1]	flows_as_client	Total number of misbehaving flows with the host as client
	flows_as_server	Total number of misbehaving flows with the host as server
unreachable_flows	flows_as_client	Total number of ICMP Port Unreachable flows with the host as client
	flows_as_server	Total number of ICMP Port Unreachable flows with the host as server
tcp_tx_stats	retransmission_packets	Total number of outgoing packets that have been retransmitted
	out_of_order_packets	Total number of outgoing packets that have been sent out-of-order
	lost_packets	Total number of outgoing packets that have been lost

[1] A misbehaving flow is defined as flow that do not meet the expected TCP behavior.

metric	name	description
contacts	num_as_client	Total number of contacts with the host as client
	num_as_server	Total number of contacts with the host as server

Table 3: Feature set used in our experiments

To extracts the data produced by ntopng we developed a software module, named *data_generator.py*, which queries directly the back-end database used by ntopng, i.e. *InfluxDB* [InfluxData, 2013], while the traffic flows through the network. Figure 4 shows the architecture used to collect the time series data.

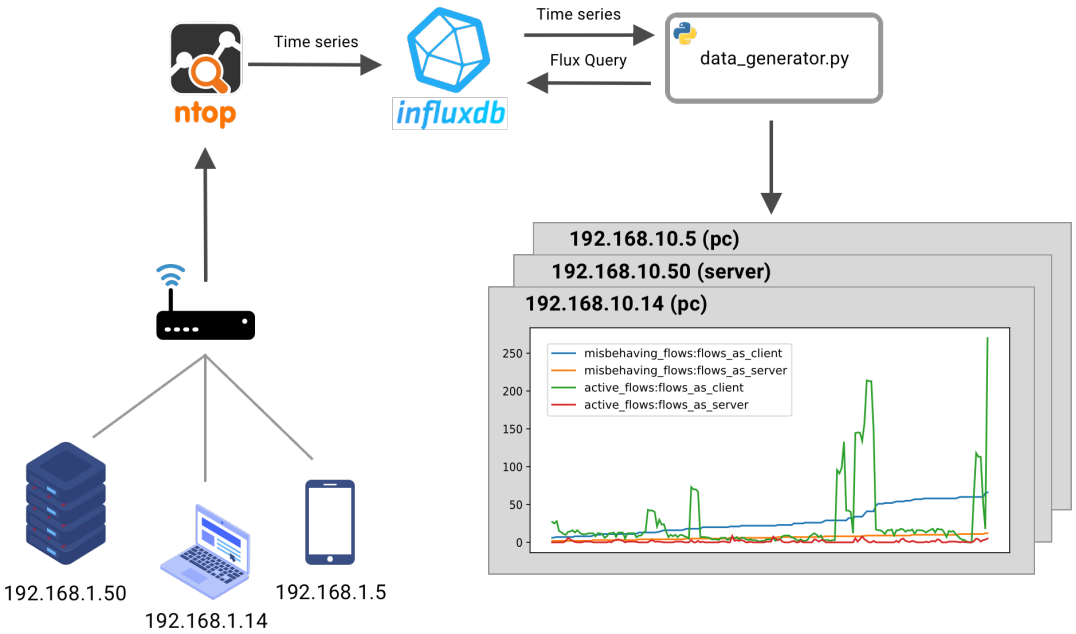

Figure 4: Multivariate time series are extracted from *InfluxDB* as the traffic flows through the network

Ntopng offers a maximum resolution of one sample every 10 seconds containing a total of 108 metrics from which the subset in table 3 is extracted. To avoid missing

values while extracting data from InfluxDB we aggregated[2] samples into windows of 15 seconds length so that at least point is surely captured in each time window. After the data has been extracted from InfluxDB, it needs preprocessing before usage. NaN values caused by time windowing [3] are filled with a rolling mean window of length 3. The data has non-stationary (non decreasing) features, like *traffic:bytes_sent* in fig. 1. We computed the difference between successive values to have a stationary time series. Furthermore we encoded the time of the day using one-hot encoding thus adding four different features: *morning, afternoon, evening, night*. We may have also used features describing if the data has been sampled on peculiar days like weekend or holidays, however the dataset used provides only midweek days.

3.2 Dataset

We conducted experiments on the *CICIDS2017* [CIC, 2017] dataset. The dataset contains the traffic captured in a network composed of fifteen hosts, servers and PCs, having as operating system one among Linux (Ubuntu), Windows (Vista, 7 Pro, 8.1) or Mac OS X. The data contains one day, Monday, of normal activity for every device in the network and four days, Tuesday to Friday, of labeled traffic containing attacks coming from an external network and executed against specific machines. The attacks are comprehensive of: *Brute Force* against *FTP* and *SSH*, *DDoS* (e.g. LOIT), *DoS* (e.g. slowloris, hulk) and port scanning. Detailed information about the attacks contained in CICIDS2017 can be found in the paper associated with the dataset: [Sharafaldin et al., 2018]. CICIDS2017 data is provided either by raw network traffic captures, i.e. pcap files, or by features extracted from each network flow using *CICFlowMeter* [Draper-Gil et al., 2016]. Thus, it has been necessary to generate the time series data using the architecture shown in figure 4, paragraph 3.1. We deployed ntopng to monitor a Linux dummy interface where the traffic has been injected using *tcpreplay* [tcp, 2017]. Tcpreplay has been essential because it allows to replay the dataset's traffic pcaps in

[2]We computed the mean value

[3]The collector queries data every 10 minutes, then the samples are aggregated in windows of 15 seconds. Depending on the time range length which can be subject to computations delays, empty micro-windows of few seconds may be returned by InfluxDB

real-time as if the machine from which we run the experiments was sniffing from the original environment. So it was possible to use ntopng as if it was deployed in the same network where the dataset was created. We used *ntopng v4.1 (CE)* to generate for each day and host a multivariate time series as described in section 3.1. A simplified representation of the dataset resulting from the data extraction process is reported in figure 5.

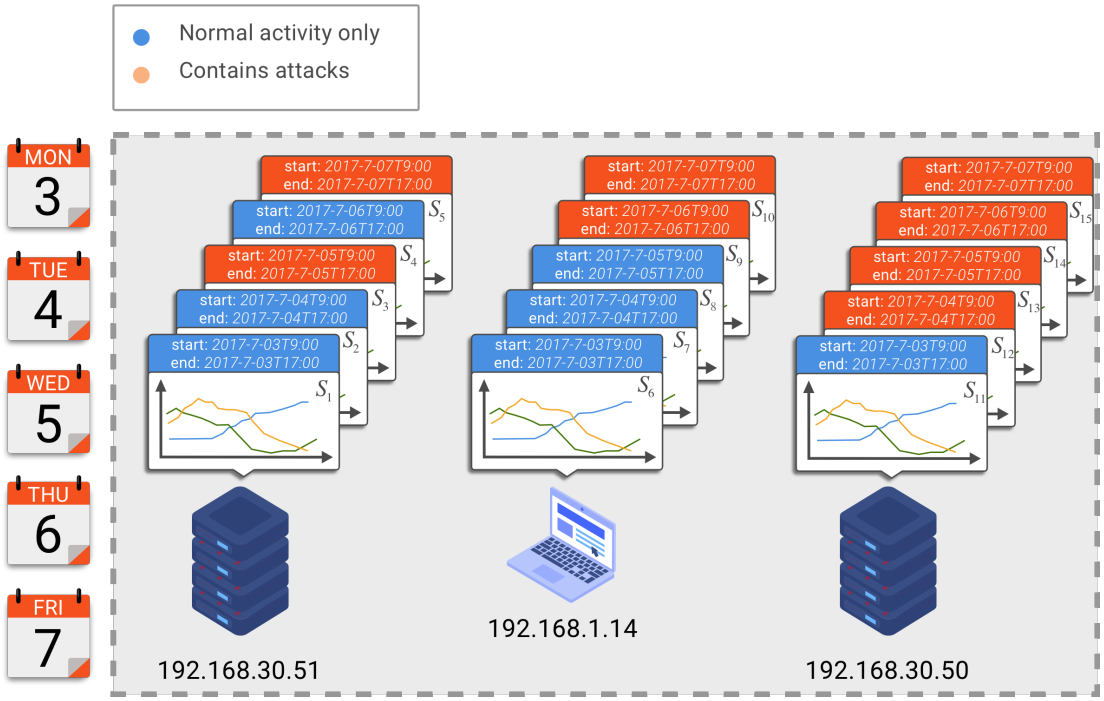

Figure 5: Graphical representation of the time series extracted from CICIDS2017. Only three of the fifteen hosts have been reported

In figure 6 we reported a more detailed example of the data extracted from ntopng for the server with IP address *192.168.10.50* before and after the preprocessing step. We have also reported, in image 7, the server network activity during two high-impact attacks (DoS and Bruteforce) to highlight a problem that we faced while analyzing the dataset. As shown in the time-series plots, the attacks labels given by the dataset don't match exactly with the attack effects on the time series. Because of the risk involved in shifting the labels of few minutes and altering the time-series, we left the labels untouched and suggest that any model that works well using this kind of labels will have better performance in real case scenario where we lack a reference label.

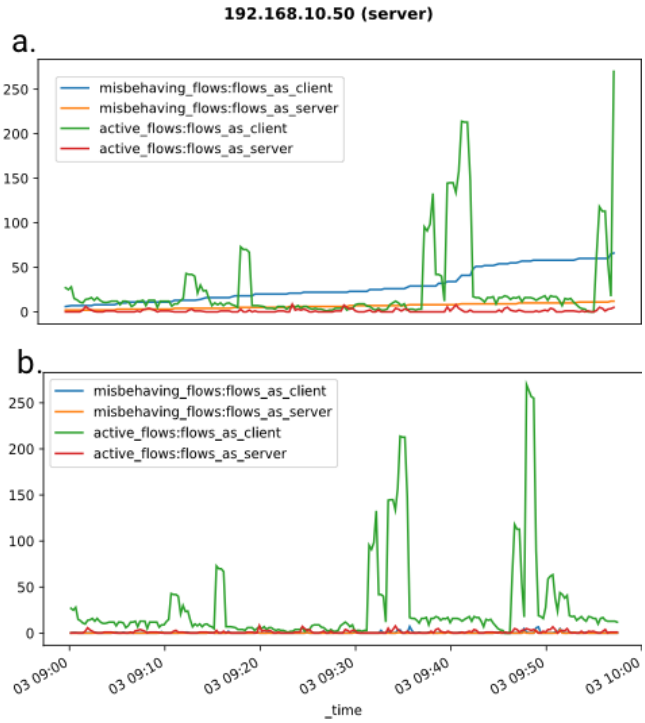

Figure 6: **a.** Network traffic time series generated by ntopng while monitoring the server 192.168.10.50 during normal activity. **b.** The multivariate time series after the preprocessing step

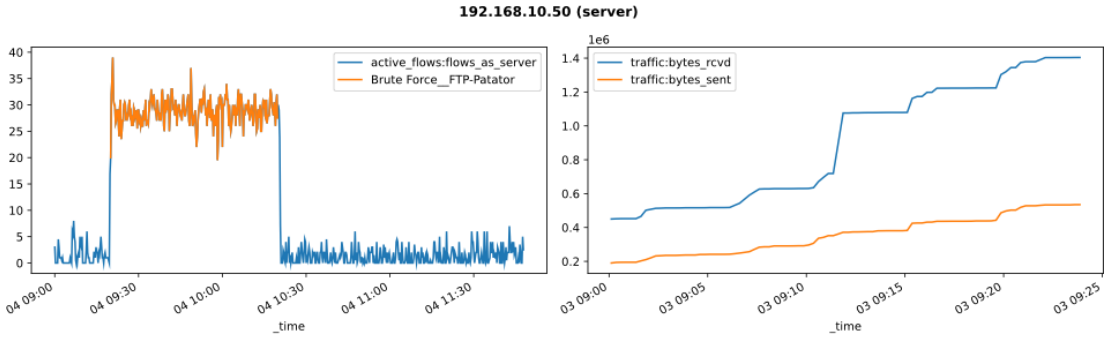

Figure 7: Time series generated by ntopng during two high-impact attacks: DoS slowloris and Bruteforce FTP-patator.

To follow the settings proposed in section 1.3 and because of the fact that our loss function is highly influenced by anomalous traffic which, by definition, is expected to be project far away from the device normal traffic in the latent space, we decided to have: *a)* a *training set* ($\mathscr{D}_{tr}$) composed of time series containing points sampled only during normal activity. The training set is used to let the model learn how normal devices behave. *b)* A *test set* ($\mathscr{D}_{m}$), containing normal activity never seen before. The

loss is tested on $\mathscr{D}_m$ at the end of the training phase. *c)* A *detection set* ($\mathscr{D}_{dt}$) containing both normal and anomalous time series generated from the network activity of previously unseen devices. Anomaly detection capabilities are tested on $\mathscr{D}_{dt}$.

To achieve the dataset subdivision into $\mathscr{D}_{tr}$, $\mathscr{D}_m$ and $\mathscr{D}_{dt}$, we separated all the time series generated for the server with IP address *192.168.10.50* from the time series generated for the rest of the network and use them as $\mathscr{D}_{dt}$. We selected the server with IP *192.168.10.50* because it is the main target for most of the attacks in the original dataset. Monday data, the one containing normal traffic only, has been used to test the model performance ($\mathscr{D}_m$). Tuesday to Friday clean data, i.e. without time-series containing attack traffic, has been used to train the model and let it learn normal devices behavior ($\mathscr{D}_{tr}$). Figure 8 summarizes the final split into $\mathscr{D}_{tr}$, $\mathscr{D}_m$ and $\mathscr{D}_{dt}$. Table 4 describes the main features of each partition.

	days	**samples**	**attacks %**	**attacks**	**hosts**
$\mathscr{D}_{tr}$	tue-fry	23172	0%	-	All except 192.168.10.50
$\mathscr{D}_m$	mon	6297	0%	-	All except 192.168.10.50
$\mathscr{D}_{dt}$	tue-fry	2087	0.127%	Brute Force (Patator) against FTP and SSH, DDoS LOIT, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS slowloris, Nmap and Portscan, brute force and XSS web attacks	192.168.10.50

Table 4: Dataset partitions with respective sizes

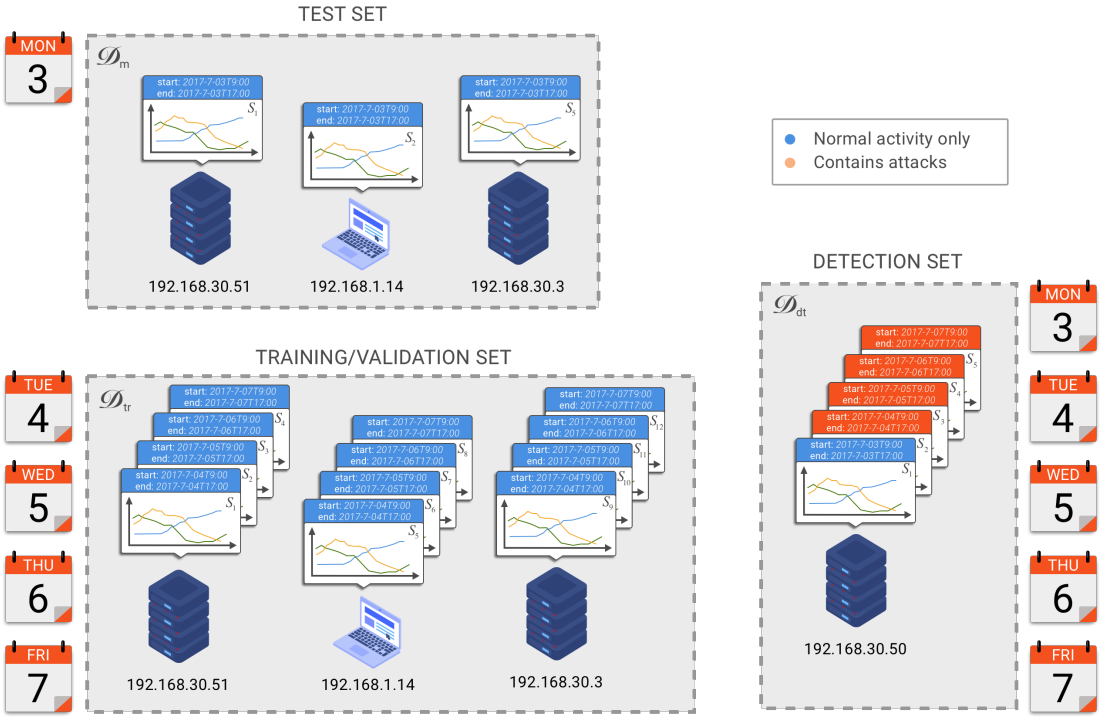

Figure 8: The training set ($\mathscr{D}_{tr}$) contains Tuesday to Friday data without attacks. The validation set ($\mathscr{D}_m$) contains Monday data only. The traffic coming from *192.168.10.50* has been used to test the detection capabilities of our system. For seek of simplicity we reported only three of the fourteen hosts in both $\mathscr{D}_{tr}$ and $\mathscr{D}_m$

After the data has been split into $\mathscr{D}_{tr}$, $\mathscr{D}_m$ and $\mathscr{D}_{dt}$, it has been preprocessed as described in section 3.1: NaN values have been filled in and differentiation has been applied to non-stationary data. After the preprocessing step, the data has been prepared to be used by our model. Each time series has been windowed in context windows of length 80, i.e. 20 minutes with 4 sample per minute. We overlapped the context windows by a factor of 95%. From now on we will take for granted that each dataset's partition contains context windows. At each time-step the model receives in input a batch of context windows from within which activities of 10 minutes (40 samples) are extracted at random. Further details are given in section 4.1.

Our training, validation and testing pipeline can be summarized into: **1.** we used 5-Fold Cross validation to train and validate our models over $\mathscr{D}_{tr}$; **2.** we trained the best configuration, i.e. the one that achieved minimum loss, on the whole $\mathscr{D}_{tr}$ and tested it over $\mathscr{D}_m$; **3.** finally, we tested the detection capabilities of our model on the previously

unseen device 192.168.10.50 ($\mathscr{D}_{dt}$) by computing metrics such as ROC area under the curve, precision and recall. Kitsune and Prophet have been trained on $\mathscr{D}_{tr}$ using the of-the-shelf *fit* functions provided by the authors with only minor modifications. More details are given in subsection 4.3.

4 Implementation details

4.1 Triplet loss

To minimize our modified triplet loss we designed a very simple architecture composed of a *GRU* followed by a last pooling layer and two fully connected layers. We used the fully connected layers' output as latent representation. At each gradient step the model is provided with a batch of context windows coming from different hosts. We extracted randomly an activity **s** from within each context **c**. Then, for each activity, the negative anchor **an** is chosen as the activity in the batch closest (using L2 norm) to **s** in the latent space but coming from a different host, as shown in figure 9. After having computed the latent representation for the host activity (e_s), the anchor positive (e_c) and the anchor negative sample (e_{an}), the three latent representations are used to compute the triplet loss and propagate the gradient backwards. In all experiment we have chosen *Adam* [Kingma and Ba, 2014] optimizer and set the loss margins α_1 and α_2 respectively to 0.01 and 0.1. That is, the distance between the activity **s** and its context **c** should be less than or equal to 0.01 and the distance between **s** and the anchor negative **an** should be further away than 0.1 in L2 terms. Both α_1 and α_2 values were chosen arbitrarily.

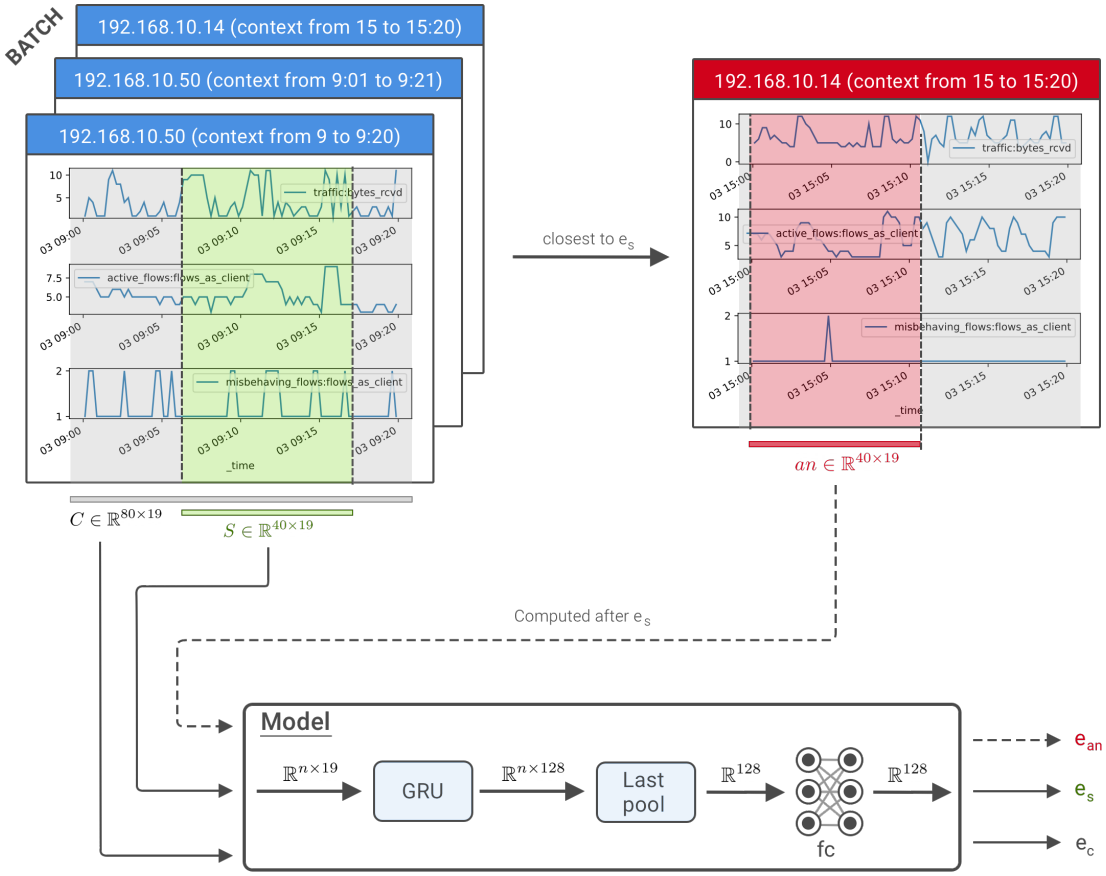

Figure 9: The model takes as input a batch of contexts at each gradient step. For each context **c**, after a fixed latent representation for activity **s** within **c** has been computed using last pooling, the negative anchor can be found in the batch. We used the letter *n* to refer to the variable sequence length: 40 samples (20 minutes) for contexts; 20 samples (10 minutes) for activities.

To compare our methods with point-wise existing ones, like *Prophet*, we computed a latent representation for each point in the multivariate time series of $\mathscr{D}_{dt}$. We used a rolling window of length 40 (i.e. an *activity*) moving from point to point and forwarding each activity in input to the model. To detect anomalous points we assumed that most of the traffic has been generated by normal activities and used density based clustering, *DB-SCAN* [Ester et al., 1996], to classified each point not belonging to the largest cluster as anomalous. We compared the results achieved using two different strategies for applying DB-SCAN. First, we tuned DB-SCAN to get the best looking clustering over the output of the t-SNE [Maaten and Hinton, 2008] dimensionality reduction technique applied over the latent representation of $\mathscr{D}_{dt}$. This resulted in look-

ing for a kinky point in the k-th nearest neighbor sorted distance plot, with k fixed a priori to 6. Fig 10 lead us to choice of $\varepsilon = 5$. We refer to this technique as TL_{t-SNE}. It is necessary to specify that we have chosen to use t-SNE as an intermediate step because it let us visualize the results and find the best clustering in a straightforward way without significantly affecting the results.

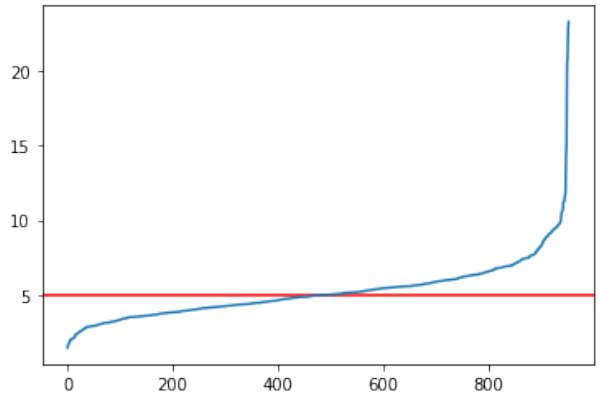

Figure 10: Distance between each point returned by the t-SNE applied to the triple loss latent representation and its 6-th nearest neighbor. We reported ε in red.

The results achieved by applying DB-SCAN to the output of t-SNE are shown in figure 15c. By comparing the clustering achieved with figure 14b it can be noted that the clustering allows us to distinguish quite accurately the normal activity from attacks' samples.

As a second alternative technique, we took advantage of the properties of the triplet loss to choose the hyper-parameters ε and *min-pts*. Indeed, ε and *min-pts* allow us to specify the density that we are looking for inside each cluster. Thus, we know a priori the density of the clusters in the latent space, or at least the one containing normal traffic. We have chosen *min-pts* to be the length of a context, i.e. 80 samples, and ε to be a value inside the range $[\alpha_1, \alpha_2]$. For seek of simplicity we have chosen:

$$\varepsilon = \frac{\alpha_1 + \alpha_2}{2}$$

We refer to this last technique as TL_{latent}. The clustering results are shown in figure 15c. It is important to note that this approach have led us to a "parameter free" unsupervised network anomaly detection technique which doesn't require an external labeled set to be tuned. Further discussion on the results achieved can be found in section 3.

4.2 Sequence to sequence autoencoder

Among the others we have chosen to compare our triplet loss based methodology with a sequence to sequence autoencoder. To this end we used a very simple architecture composed of two single layer *GRUs* which act as encoder and decoder module and a third two layer fully connect network which reconstruct the original multivariate sample from the decoder's output at each time step. The encoder's hidden state is initialized with an empty full-zero vector. The encoder's output is used to create a fixed-size latent representation using mean or last pooling. The latent representation is then used to initialize the decoder module hidden state, whereas the decoder first input is always a predefined token: *[INIT]*. We trained the autoencoder to reconstruct the original input signal by minimizing the mean squared error over each time series. In each experiment we have chosen to use Adam optimizer and context windows of length 15 (nearly 3 minutes) with overlapping of 0.95%. The training framework is identical to the one shown in figure 9, except for the reduced context length and the fact that the model takes as input a full context $\mathbf{X}$ and gives in output a reconstructed context $\hat{\mathbf{X}}$. Both are then used to compute the *mean squared error* at each step. We validated our models with different percentages of teacher forcing and chosen the configuration that achieved minimum reconstruction error. Figure 11 briefly summarizes the architecture used as sequence to sequence autoencoder.

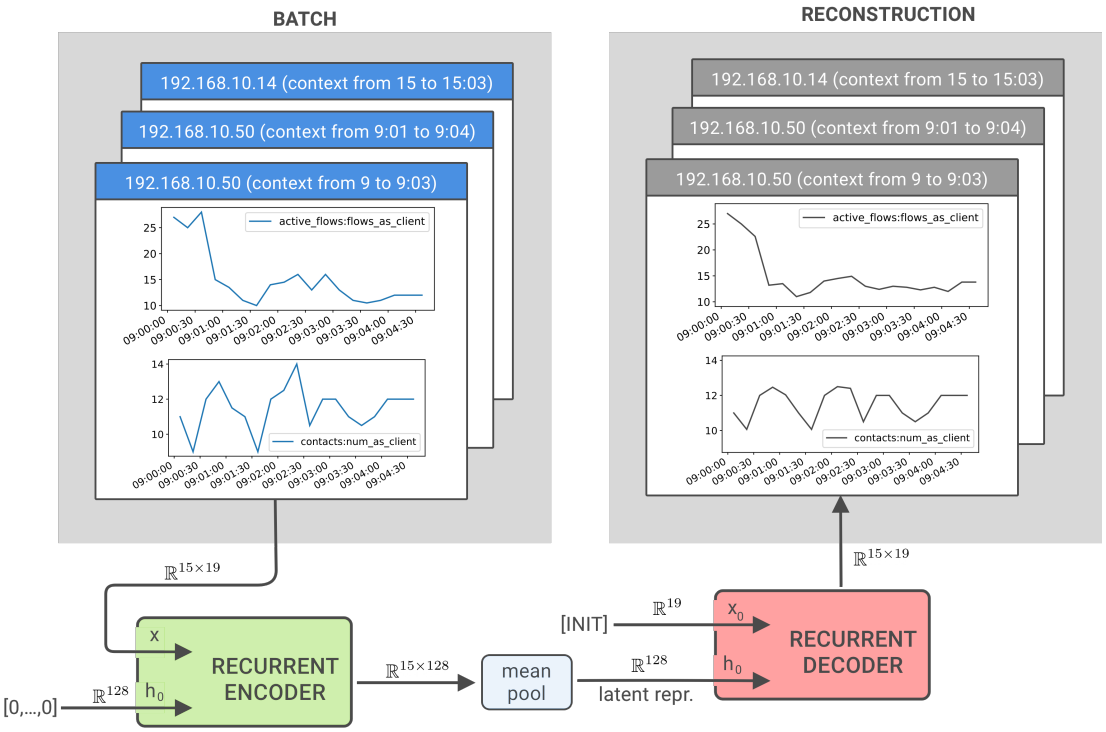

Figure 11: The autoencoder receives in input a batch of multivariate time series and encodes them into a latent, fixed size representation. The latent representation is used as hidden state for a decoder module.

To detect anomalous points we applied *DB-SCAN* as for the triple loss case. We computed the latent representation for each point in the time series using a sliding window of length 15 moving from point to point. We applied t-SNE algorithm to the latent representation of each time series, then we set a priori the *min-samples* DB-SCAN's hyperparameter to 5 and chosen ε by looking at the sorted distances plot between each point and its 5-th nearest neighbor, figure 12, which led us to the choice of $\varepsilon = 5$. We assumed that most of the traffic is benign, thus we classified as anomalous all the points not belonging to the largest cluster in the t-SNE space. We may have also chosen to use DB-SCAN in the latent space without applying a dimensional reduction technique, however we found much more controllable and straightforward to tune DB-SCAN and detect anomalies using such technique. Furthermore no significant changes in the detection capabilities has been noted. The results achieved are shown in figure 15a-b.

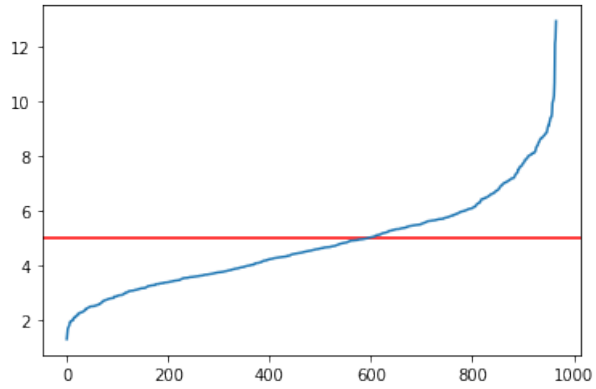

Figure 12: Distance between each point returned by the t-SNE applied to the autoencoder latent representation and its 5-th nearest neighbor. We reported ε in red.

4.3 Kitsune and Prophet

We used the official off-the-shelf implementation of Kitsune [ymirsky, 2018] to test its detection capabilities. No significant changes have been made except the implementation of the detection threshold according to the original paper [Mirsky et al., 2018], which can be summarized in: when a new instance arrives, raise an alert if the computed root mean square reconstruction error exceeds the maximum training error weighted by a sensitivity parameter β. We tuned β on $\mathscr{D}_{dt}$ to achieve the maximum performance using a grid search.

Due to the fact that Prophet doesn't support multi-output predictions, we trained multiple instances of Prophet for each channel in the time series of $\mathscr{D}_{tr}$ independently, giving to each all the rest of the channels to learn from. Assume that $\mathscr{D}_{tr}$ contains a single multivariate time series $C = \{c_1, ..., c_k\}$, with k channels. We trained k instances of Prophet, each one to target a single channel. For channel i we trained the Prophet instance P_i using the channels set $C_i = \{c_j | j \in [1, .., k] \wedge j \neq i\}$. After having trained $P_1, ..., P_k$ we applied each instance to predict the values of the corresponding channel c_i using the values in C_i. For each point in $\mathscr{D}_{dt}$ we raised an anomaly if at least half of the Prophet predictors were wrong, i.e. out of the upper or lower boundary. A last detail needs to be clarified. At each timestamp, Prophet requires a single value for each channel, i.e. a single time series generated by a single source. Instead, in $\mathscr{D}_{tr}$, for each

sample in time we have multiple values for each channel, specifically one for each host. That is, we have m multivariate time series, with m being the number of hosts. Due to the fact that we only have four days and thus our data has at most daily seasonality features, we shifted the time series generated for each host to different years. This let Prophet learn from $\mathscr{D}_{tr}$ as if it were a single multivariate time series generated for four days of different years, with each year containing the traffic of a specific host in the network. Figure 13 shows how $\mathscr{D}_{tr}$ has been changed to match Prophet expected input.

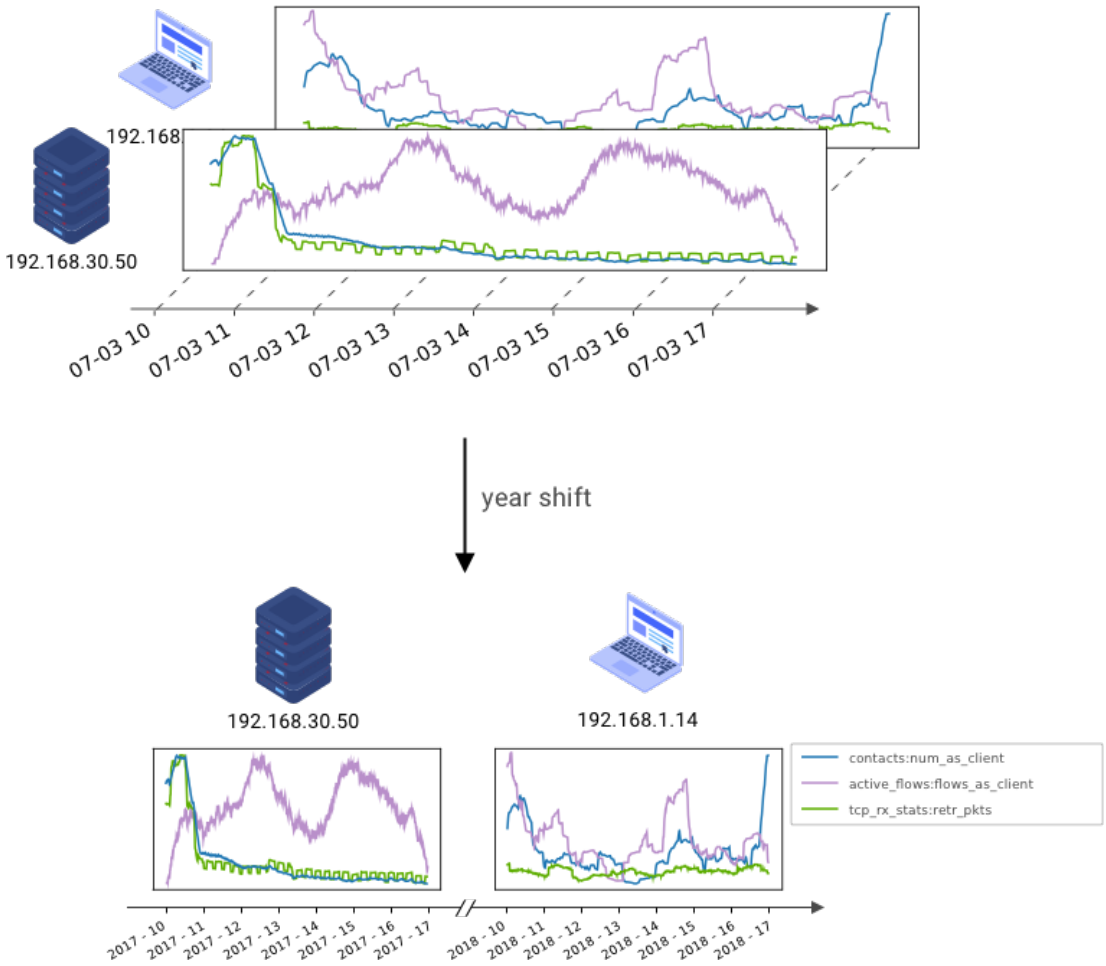

Figure 13: To match Prophet input format we shifted the time series of different devices to different years.

5 Results

We executed a grid search over $\mathscr{D}_{tr}$ to tune the main parameters of our triplet loss based model (*TL*): learning rate (*lr*); fully connected input and output number of neurons (*latent size*); number of recurrent layers (*rnn layers*); GRU's number of neurons (*rnn size*). We fixed the batch size for the triplet loss at 4096 context windows. We also experimented different configurations for the autoencoder (*AE*) by tuning: learning rate; batch size; pooling (mean or last); teacher forcing ratio; GRU's hidden layer and fully connect layers' size (*latent size*). For every configuration we used the optimizer Adam and applied 5-Fold cross validation to compute the mean validation loss (*mean vl*). At each step we used *early stopping* on the loss value computed over a validation partition (0.2%) of the training set fold at hand. Table 5 and 6 reports the result achieved. With the best hyper parameter configuration (marked in bold), we trained both *AE* and *TL* on the full training set, $\mathscr{D}_{tr}$, and tested the detection capabilities on $\mathscr{D}_{dt}$. For each technique we computed: *precision (prec.), recall (rec.), accuracy (acc.)*, ROC area under the curve (*roc auc*) and computed the whole *confusion matrix*. We reported the results achieved in table 7 together with the results achieved by *Kitsune*, also trained on $\mathscr{D}_{tr}$. We tuned Kitsune sensitivity hyperparameter, β in the original paper, the maximum size of any autoencoder in the ensemble layer, *m* in the authors' code, and reported the configuration which achieved the best detection performance. We specified the technique used to fit DB-SCAN using the subscript: TL_{t-SNE}, meaning that DB-SCAN has been fit on the output of t-SNE; TL_{latent}, meaning DB-SCAN has been fit over the latent space using a priori knowledge resulting from the use of the triplet loss itself, as described in section 4.1. The results attained by Prophet have not been reported because, according to our predictions in section 1.3, the tool achieved a very poor detection performance ($ROC = 0.5$). We argue that this is due to the fact that Prophet is not able to combine the information coming from time-series generated by different hosts and apply it to previously unseen ones if these are significantly different.

lr	latent size	rnn layers	rnn size	mean vl
0.0005	**128**	**3**	**128**	**0.0820**
0.0005	128	3	64	0.0822
0.0005	64	3	128	0.0841
0.0001	64	3	128	0.0849
0.0001	64	3	64	0.0852

Table 5: Grid search results achieved with 5-Fold cross validation on D_{tr} using triplet loss based architecture with context length of 80 (20 minutes), overlapping of 0.95 and last pool.

batch size	lr	latent size	teacher forcing ratio	mean vl
64	**0.0001**	**128**	**0.7**	**1151.5249**
128	0.0001	128	1.	1152.8282
64	0.0001	32	1.	1161.3983
128	0.0001	32	1.	1174.2008
64	0.0001	64	1.	1193.9202

Table 6: Grid search results achieved with 5-Fold cross validation on D_{tr} using sequence to sequence autoencoder based architecture with context length of 15 (3 minutes), overlapping of 0.95. Pooling has been omitted because mean pool has always led to better results.

	roc auc	prec.	rec.	acc.	f1	tn	fp	fn	tp
AE	**0.868**	**0.72**	**0.792**	**0.921**	**0.754**	**0.801**	**0.046**	**0.031**	**0.12**
TL_{t-SNE}	0.82	0.846	0.663	0.929	0.744	0.827	0.018	0.051	0.102
TL_{latent}	0.839	0.695	0.738	0.909	0.716	0.796	0.049	0.04	0.113
Kitsune	0.638	0.556	0.322	0.858	0.408	0.81	0.038	0.102	0.048

Table 7: Detection capabilities of different approaches tested on D_{dt}.

Besides the fact that the best scores are achieved by the sequence to sequence autoencoder architecture, we point out that by using a priori knowledge given by the triplet loss to fix the DB-SCAN hyperparameters (TL_{latent}) has led to similar results. Kitsune achieves some results but it is not comparable with time-aware methods. This is probably due to the considerable noise caused by the chaotic nature of the network data when analysed point by point.

We explored the latent space by examining the scatter plot generated by applying t-SNE on the latent representation of each sample in $\mathscr{D}_{dt}$. Figure 14 shows the input space without having applied any learning algorithm, the latent space resulting from the minimization of the triplet loss, the latent representation generated by sequence to sequence autoencoders.

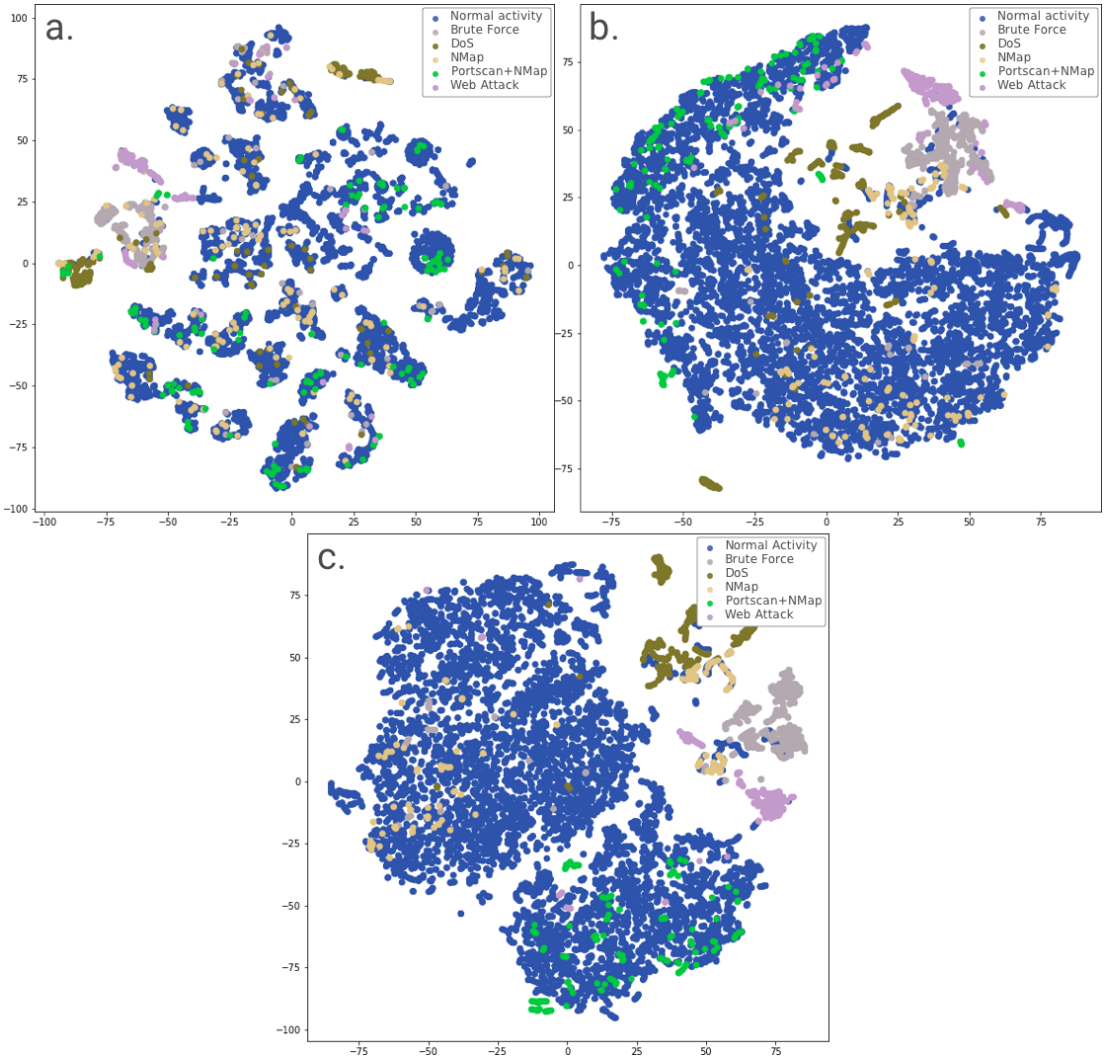

Figure 14: All the plots have been generated using data from $\mathscr{D}_{dt}$. **a.** t-SNE applied to the raw input samples. **b.** t-SNE applied over the latent representation resulting from the minimization of the triplet loss. **c.** t-SNE applied on the latent representation built by the sequence to sequence autoencoder.

We highly suspect that the noise within the t-SNE clusters of figure 14 and a degraded performance is caused by the time shift described in section 3.1, which consists of having labels wrongly shifted from network effects of the attacks.

We also reported, in figure 15, the clusters used to detect anomalies (recall that we considered as anomalous each point not belonging to the largest cluster, see section 4.1 and 4.2). We point out: a) the clusters contain similar attacks, meaning anomalies that have similar effects on the time series end up being near in the latent space; b)

Our a priori hyperparameter choice for DB-SCAN over the latent representation of *TL* gives us a very good choice for detecting anomalies.

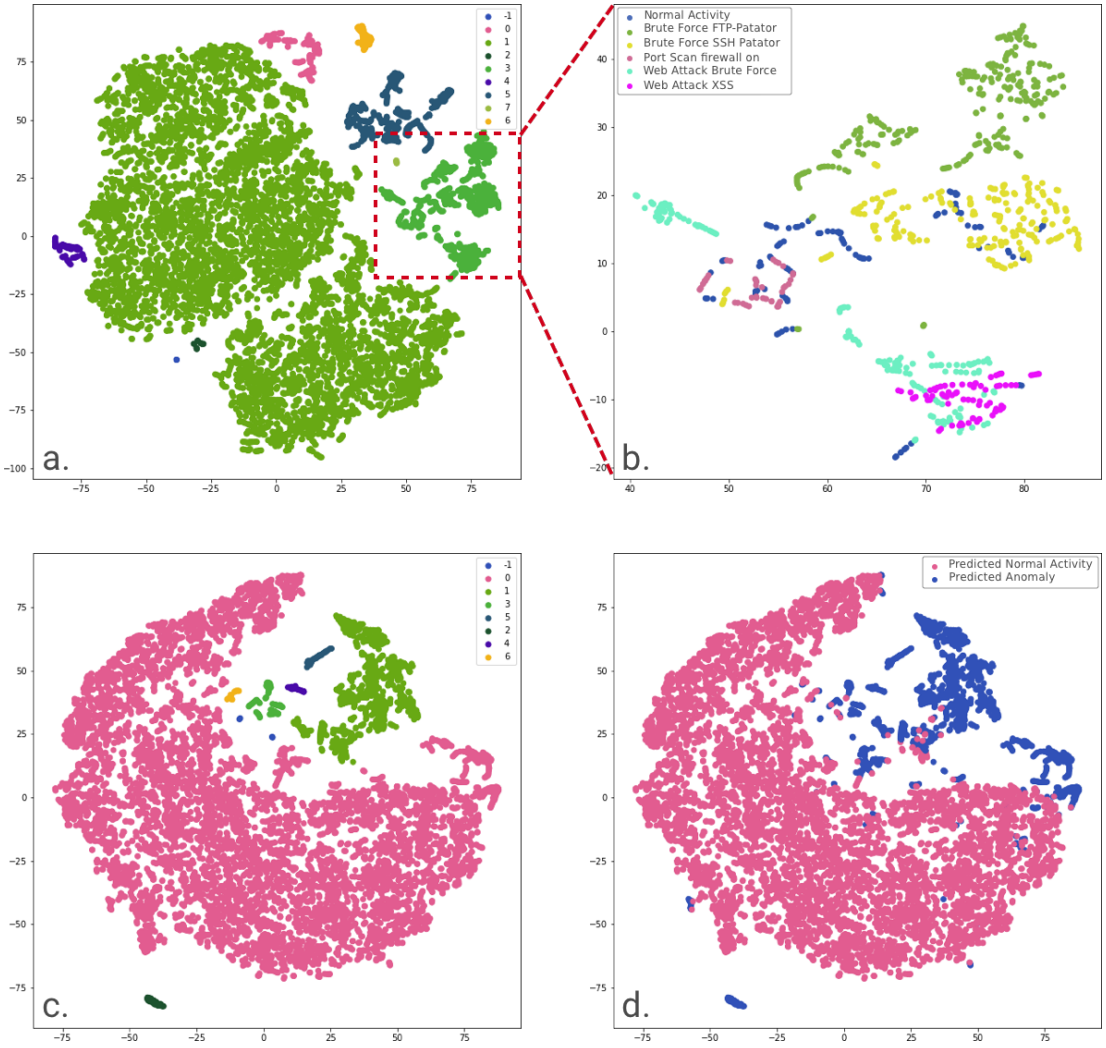

Figure 15: Plots generated using the data from $\mathscr{D}_{dt}$. **a.** t-SNE applied on the latent representation built by the sequence to sequence autoencoder. **b.** DB-SCAN applied to the t-SNE output of the latent representation generated by sequence to sequence autoencoder; **c.** focus on the cluster with label "3".

We have also plotted the output from the *t-SNE* reduction applied to the latent representation of $\mathscr{D}_m$. From our experiments the latent representation of similar device cluster together both for autoencoders and triplet loss, as shown in figure 16. Figure 16 has also given us an insight about an unexpected device not reported by the authors of the dataset, with IP 192.168.10.1, which forms a well separated cluster labeled

"unknown device class".

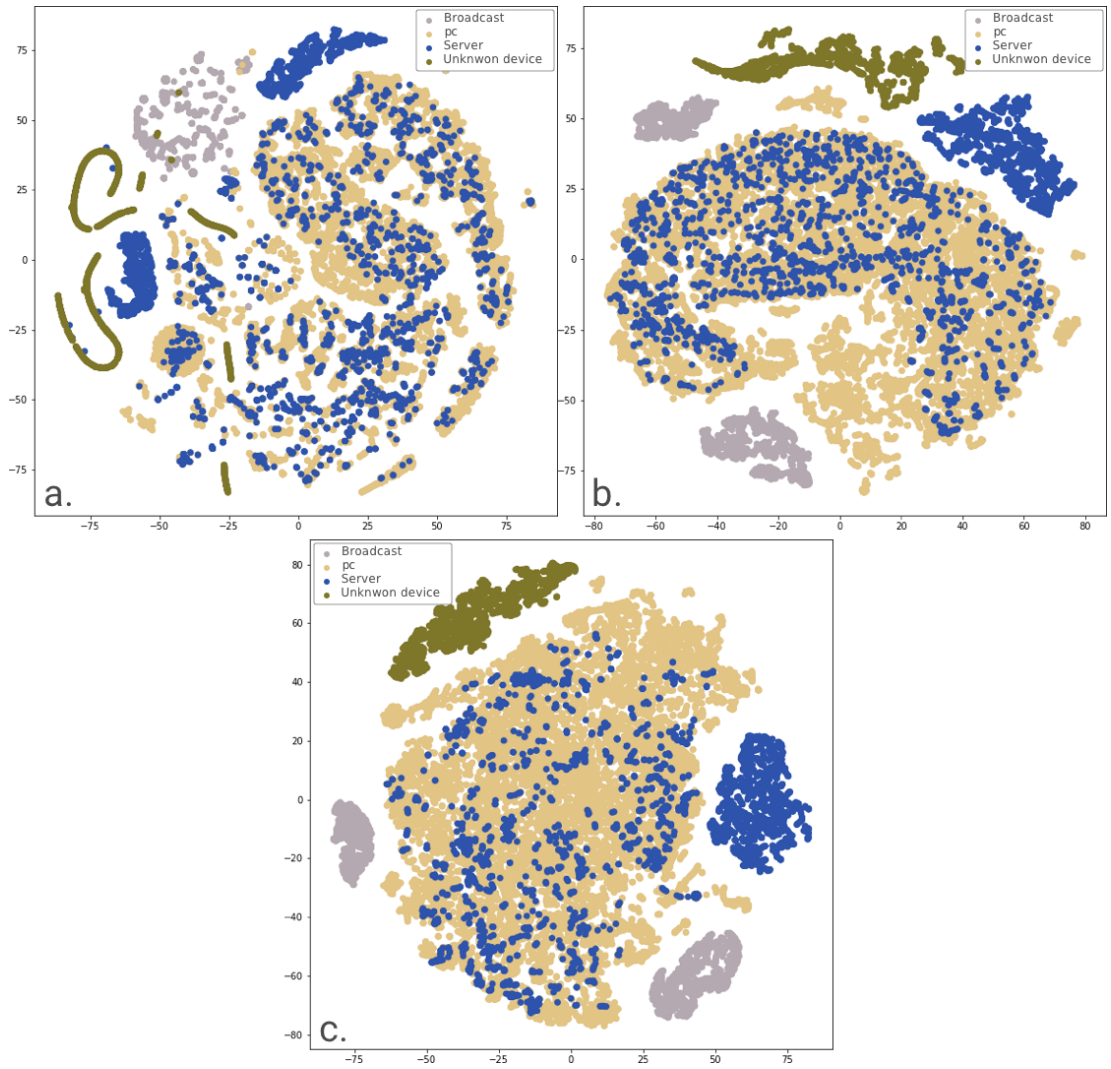

Figure 16: Plots generated using the data from $\mathscr{D}_m$. **a.** t-SNE applied to the raw input samples. **b.** t-SNE applied on the latent representation resulting from the minimization of the triplet loss. **c.** t-SNE applied on the latent representation built by the sequence to sequence autoencoder.

As a last experiment we merged $\mathscr{D}_m$ with a subset of the *IoT23* dataset, presented in [Parmisano et al., 2020] and containing *internet of things (IoT)* benign and malicious traffic. The scenarios of IoT23 that we used are: normal traffic captured from an Amazon Echo device and a Soomfy IoT doorlock; Mirai botnet [jgamblin, 2017] malicious traffic and a Trojan attack (scenario 8 in the dataset authors repository).

Both the attacks have been executed against a RaspberryPI for which the authors didn't provide benign traffic. Figure 17 shows the results achieved by applying t-SNE to the latent representation of our triplet loss based model pretrained on $\mathscr{D}_{tr}$. Similar but less good results were achieved using the pretrained sequence to sequence autoencoder. The data available is not sufficient to outline a typical IoT behavior in the latent space and consequently discern benign traffic from malicious one. However it is possible to see that the model has generalized well and the latent space is much more structured than the raw data and able to fully characterize the traffic captured from two previously unseen IoT devices: the Amazon Echo and the Smoofy doorlock.

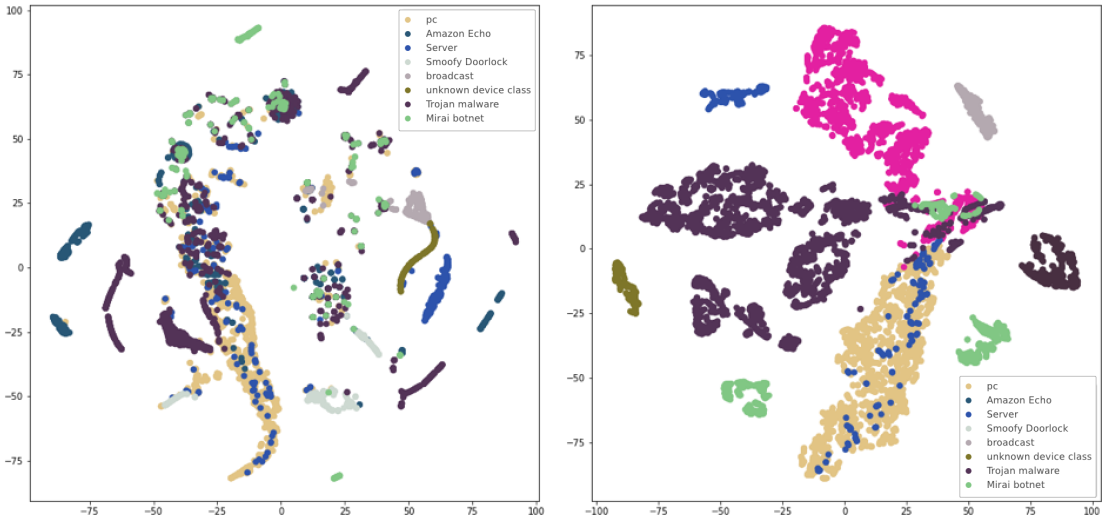

Figure 17: Plots generated using the data from $\mathscr{D}_m$ merged with IoT23 data. **Left:** t-SNE applied to the raw input samples. **Right:** t-SNE applied on the latent representation resulting from the minimization of the triplet loss.

6 Conclusion

In our work we highlighted the use of the feature space latent representation to improve network monitoring. Through the use of unsupervised learning and dimensionality reduction techniques we were able to visualize the network activity generated by different devices and introduce the concept of network behavior similarity among different hosts. Using prior knowledge on the latent space density resulting from the use of a

triplet loss objective function led us to a threshold-free unsupervised anomaly detection system which achieves performance comparable to sequence to sequence autoencoders, one of the most popular techniques. From our experiments, whose source code has been publicly released [Sabella, 2021], we found that time-aware methods achieve better performance than point-wise techniques like the one proposed by [Mirsky et al., 2018]. We argue that this is due to the inherent noise in network activity. We proved that simpler techniques like auto-regressors are unable to combine the knowledge acquired from different hosts. Therefore, we suggest that they are not suitable for a scenario that includes learning from many captures from multiple uninfected devices and detecting anomalies for previously unseen hosts.

From our results both sequence to sequence autoencoders and triplet loss are to be considered worth of further study. The former because of their performance, the latter because of the deeper understanding and better control over the latent representation. Further experiments are also needed to asses the performance of the latent representation achieved by multi step ahead neural network objectives.

With our work we point out, as done previously by [Hindy et al., 2020], that the modern datasets are not sufficient to implement a reliable anomaly detection system. We proposed an unsupervised learning framework which consists of training on raw traffic captured from different and diverse devices and testing detection performance on a small dataset containing malicious traffic captured from few devices. If future studies will prove that it is possible to build a strong latent representation using unsupervised models even with anomalous traffic in the training dataset, it may be possible to collect a training set containing enough information in terms of volume and diversity able to enhanced the results achieved by our experiments.

The models proposed are far from being perfect, still many questions and problems remain open.

References

[CSE, 2009] (2009). NSL-KDD dataset. Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC).

[CIC, 2017] (2017). Cicids2017. *URL https://www.unb.ca/cic/datasets/ids-2017.html*. Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC).

[tcp, 2017] (2017). tcpreplay. *URL https://tcpreplay.appneta.com/*. aa.

[Aygun et al., 2017] Aygun, Can, R., and Yavuz, A. G. (2017). Network anomaly detection with stochastically improved autoencoder based models. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 193–198. IEEE.

[Banville et al., 2019] Banville, H., Moffat, G., Albuquerque, I., Engemann, D.-A., Hyvärinen, A., and Gramfort, A. (2019). Self-supervised representation learning from electroencephalography signals. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

[Casas et al., 2011] Casas, Pedro, Mazel, J., and Owezarski, P. (2011). Unada: Unsupervised network anomaly detection using sub-space outliers ranking. In *International Conference on Research in Networking*, pages 40–51. Springer.

[Chalapathy et al., 2019] Chalapathy, Raghavendra, and Chawla, S. (2019). Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*.

[Chen et al., 2002] Chen, Ning, Chen, A.-z., and Zhou, L.-x. (2002). An incremental grid density-based clustering algorithm. *Journal of software*, 13(1):1–7.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Draper-Gil et al., 2016] Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I., and Ghorbani, A. A. (2016). Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414.

[Dromard et al., 2019] Dromard, Juliette, Owezarski, and Philippe (2019). Study and evaluation of unsupervised algorithms used in network anomaly detection. In *Proceedings of the Future Technologies Conference*, pages 397–416. Springer.

[Dromard et al., 2016] Dromard, Juliette, Roudière, Gilles, Owezarski, and Philippe (2016). Online and scalable unsupervised network anomaly detection method. *IEEE Transactions on Network and Service Management*, 14(1):34–47.

[Ergen et al., 2019] Ergen, Tolga, and Kozat, S. S. (2019). Unsupervised anomaly detection with lstm neural networks. *IEEE transactions on neural networks and learning systems*.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

[Franceschi et al., 2019] Franceschi, J.-Y., Dieuleveut, A., and Jaggi, M. (2019). Unsupervised scalable representation learning for multivariate time series. In *Advances in Neural Information Processing Systems*, pages 4652–4663.

[Hettich, 1999] Hettich, S. (1999). Kdd cup 1999 data. *The UCI KDD Archive*.

[Hindy et al., 2020] Hindy, H., Brosset, D., Bayne, E., Seeam, A., Tachtatzis, C., Atkinson, R., and Bellekens, X. (2020). A taxonomy of network threats and the effect of current datasets on intrusion detection systems. *IEEE Access*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[InfluxData, 2013] InfluxData (2013). influxdb. URL https://github.com/influxdata/influxdb.

[jgamblin, 2017] jgamblin (2017). Mirai-source-code. URL https://github.com/jgamblin/Mirai-Source-Code.

[Kathareios et al., 2017] Kathareios, Georgios, Anghel, A., Mate, A., Clauberg, R., and Gusat, M. (2017). Catch it if you can: Real-time network anomaly detection with low false alarm rates. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 924–929. IEEE.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kwitt et al., 2007] Kwitt, Roland, and Hofmann, U. (2007). Unsupervised anomaly detection in network traffic by means of robust pca. In *2007 International Multi-Conference on Computing in the Global Information Technology (ICCGI'07)*, pages 37–37. IEEE.

[Lin et al., 2018] Lin, Zilong, Shi, Y., and Xue, Z. (2018). Idsgan: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077*.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Malhotra et al., 2015] Malhotra, P., Vig, L., Shroff, G., and Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89. Presses universitaires de Louvain.

[Mirsky et al., 2018] Mirsky, Y., Doitshman, T., Elovici, Y., and Shabtai, A. (2018). Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*.

[Mirza et al., 2018] Mirza, H, A., and Cosan, S. (2018). Computer network intrusion detection using sequential lstm neural networks autoencoders. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE.

[Moustafa et al., 2019] Moustafa, Nour, Hu, J., and Slay, J. (2019). A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, 128:33–55.

[ntop, 2017] ntop (2017). ntopng. URL https://github.com/ntop/ntopng.

[Parmisano et al., 2020] Parmisano, A., Garcia, S., and Erquiaga, M. (2020). A labeled dataset with malicious and benign iot network traffic. *Stratosphere Laboratory: Praha, Czech Republic*.

[Pena et al., 2013] Pena, E. H., de Assis, M. V., and Proença, M. L. (2013). Anomaly detection using forecasting methods arima and hwds. In *2013 32nd International Conference of the Chilean Computer Science Society (SCCC)*, pages 63–66. IEEE.

[Rogers and McClelland, 2014] Rogers, T. T. and McClelland, J. L. (2014). Parallel distributed processing at 25: Further explorations in the microstructure of cognition. *Cognitive science*, 38(6):1024–1077.

[Sabella, 2021] Sabella, S. (2021). Detecting network anomalies. URL https://github.com/samuelesabella/Detecting-network-anomalies-using-the-feature-space-latent-representation.

[Schölkopf et al., 2001] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

[Sharafaldin et al., 2018] Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116.

[Sommer et al., 2010] Sommer, Robin, and Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE.

[Tax and Duin, 2004] Tax, D. M. and Duin, R. P. (2004). Support vector data description. *Machine learning*, 54(1):45–66.

[Taylor and Letham, 2018] Taylor, S. J. and Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1):37–45.

[Usama et al., 2019] Usama, Muhammad, Qadir, J., Raza, A., Arif, H., Yau, K.-L. A., Elkhatib, Y., Hussain, A., and Al-Fuqaha, A. (2019). Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE Access*, 7:65579–65615.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).

[Winters, 1960] Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342.

[Yaacoubi, 2019] Yaacoubi, O. (2019). The rise of encrypted malware. *Network Security*, 2019(5):6–9.

[ymirsky, 2018] ymirsky (2018). Kitnet. URL https://github.com/ymirsky/KitNET-py.

[Zhou et al., 2005] Zhou, B., He, D., Sun, Z., and Ng, W. H. (2005). Network traffic modeling and prediction with arima/garch. In *Proc. of HET-NETs Conference*, pages 1–10.