# Time-based Anomaly Detection using Autoencoder

Mohammad A. Salahuddin[1], Md. Faizul Bari[1], Hyame Assem Alameddine[1,2],
Vahid Pourahmadi[1], and Raouf Boutaba[1]
[1]David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
[2]Ericsson Security Research, Montreal, Canada
{mohammad.salahuddin, faizul.bari, halamedd, v2pourah, rboutaba}@uwaterloo.ca

***Abstract*—Distributed Denial of Service (DDoS) attacks continue to draw significant attention, especially with the recent surge in cyber attacks that targeted the healthcare, education and financial sectors, during the COVID-19 pandemic. The expansion of virtualization and softwarization technologies, and the surge in Internet of Things (IoT) devices, increase the attack surface and the impact of attacks on networks. In this paper, we present a novel time-based anomaly detection system that leverages an Autoencoder. We explore the impact of different time-windows on detecting multiple DDoS attacks that are difficult to detect via the widely used flow-based features. We train and evaluate our Autoencoder on the recent CICDDoS2019 dataset, and show that our approach achieves an anomaly detection F1-score of over 99% for most attacks and greater than 95% for all attacks.**

***Index Terms*—Security management, distributed denial of service, anomaly detection, autoencoder**

## I. Introduction

A number of unique cyber-crimes recently took place following the unprecedented COVID-19 pandemic, which impacted society and businesses [1]. Cyber-criminals took advantage of the pandemic to expand upon their arsenal, in the light of an increased reliance on telecommunication networks. Multiple Distributed Denial of Service (DDoS) attacks took place recently, which targetted the healthcare, financial, government sectors, food services industry and the general public [1], [2]. An example is the DDoS attack on OKEx and Bitfinex financial institutions.

DDoS attacks can deplete network resources by increasing network traffic and prevent legitimate users from accessing the network [3], [4]. DDoS attacks take advantage of existing vulnerabilities in virtualization technologies (*e.g.*, virtual machines, containers, *etc*.) and Internet of Things (IoT) devices, which can be harnessed as part of a botnet to launch attacks [4], [5]. DDoS attack detection techniques can be broadly classified into: (i) signature-based approaches for detecting known attacks [5]–[10], and (ii) anomaly-based approaches that can detect both one-day, as well as zero-day attacks [11].

Anomaly detection consists of establishing a baseline of normal behavior of the protected system and identifying any deviations from the norm [11]. Machine Learning (ML) has been widely adopted in the literature to address anomaly detection [12]. Supervised learning has been employed in [13], [14] to detect anomalies. While supervised learning techniques provide high accuracy [15], they depend on the availability of a labeled dataset that distinguishes normal from anomalous instances. As anomalous instances are usually fewer than normal, these methods face challenges related to dataset imbalance. In addition, obtaining labeled anomalous traffic is difficult, given that attack training data is very rare [11], [16]. In contrast, unsupervised learning techniques [6], [15]–[18] do not require labeled data and assume that benign data adopts similar behavior. Deviation from the profiled behavior is then detected as anomalous [11], [16]. Neural networks have been used for anomaly detection, due to their ability to learn complex and non-linear relationships in input data, resulting in high detection rates. In contrast, traditional ML approaches suffer from high false alarms on large datasets and require experts for proper feature selection [4], [6].

We present a time-based anomaly detection system that leverages an Autoencoder, a neural network with two components—encoder and decoder. It compresses the input data into a latent low-dimensional space through the encoder, and decompresses the compressed data to reconstruct the original input using the decoder, while minimizing the reconstruction error [16], [17]. An Autoencoder learns the data representation in an unsupervised manner, and can be efficiently used to profile benign network traffic [16]. While most works employ flow-based statistical features with ML models to detect anomalies, we highlight the shortcomings of flow-based features in detecting specific DDoS attacks. We evaluate the impact of time-windows in detecting numerous DDoS attacks by leveraging time-based features. Time-based features depict statistical information of a subset of packets collected over a specific period or time-window. We evaluate our proposed approach using the CICDDoS2019 dataset [19]. The major contributions of this paper are:

- We develop a novel time-based anomaly detection system that leverages an Autoencoder for efficient DDoS detection.
- We explore the impact of multiple time-windows and their aggregation on the performance of detecting anomalous DDoS traffic.
- We showcase the robustness of our anomaly detection system to zero-day attacks, which is primarily attributed to time-based features.
- Our system achieves high F1-scores in detecting TCP SYN, UDPLag, NetBIOS, and PortMap attacks, which are difficult to detect via flow-based features.

978-3-903176-31-7 

The rest of the paper is organized as follows. Section II provides a literature review. Section III presents our time-based anomaly detection system, while Section IV exposes the CICDDoS2019 dataset. Our experimental results and analysis are discussed in Section V. In Section VI, we conclude with a brief summary and instigate future research directions.

## II. Literature review

### A. Machine learning for anomaly detection

Doshi *et al.* [13] study DDoS attack detection in an IoT environment. They test 5 different classification algorithms, including K-nearest neighbors (KNN), Support Vector Machine with linear kernel (L-SVM), Decision Tree (DT), Random Forest (RF) and Neural Network (NN). The authors show that these algorithms achieve an accuracy of over 99%. They employ stateless features derived from flow characteristics for individual packets, along with stateful statistical features collected over a time-window of 10 seconds. Their work is limited to detecting only three types of DDoS attacks *i.e.*, TCP SYN, UDP flood and HTTP GET flood attacks. Sarraf *et al.* [14] use the CICIDS2017 dataset to train a DT and a L-SVM for DDoS attack detection. They achieve an accuracy of close to 100%. The above works consider supervised learning techniques, which require labeled anomalous traffic that is difficult to obtain [11], [16].

To overcome the supervised learning shortcomings, unsupervised learning has been employed. Sharafaldin *et al.* [19] generate the CICDDoS2019 dataset that encompasses 13 different DDoS attacks. They evaluate the performance of 4 different classical ML techniques, including ID3, RF, Naïve Bayes, and logistic regression, in detecting the different attacks. The authors employ flow-based statistical features extracted using CICFlowMeter [20]. However, their reported results show poor detection performance (*i.e.*, low F1 score). Elsayed *et al.* [4] develop an intrusion detection system against DDoS attacks in a SDN environment, based on Recurrent Neural Network (RNN) with an autoencoder. The authors argue that deep learning algorithms with sequential traffic cause loss in data information. Hence, they employ RNN to address this shortcoming, and use flow-based features from CICFlowMeter using the CICDDoS2019 dataset.

Choi *et al.* [17] leverage the NSL-KDD dataset to train and test different architectures of Autoencoder. They develop a heuristic to determine a threshold of reconstruction error and report that their unsupervised anomaly detection technique outperforms other clustering algorithms. The authors in [16] develop an Autoencoder for DDoS attack detection, which is trained and tested on public and synthetic datasets. They consider sub-flow features to reduce the response time of DDoS. They show that their model achieves 82% detection rate based on their reconstruction error threshold selection, which is tailored to achieve zero false positives. Their Autoencoder performs better than other traditional ML algorithms. Most of the above works leverage flow-based features, which are only able to detect a subset of network attacks as they are oblivious to packet-level information [7], [21], [22].

Mirsky *et al.* [6] propose Kitsune, that leverages statistical temporal features for different time-windows to capture the behavior of a packet's channel (*i.e.*, conversation). Kitsune is a plug-and-play network intrusion detection system designed to be light weight, for deployment on any low memory and processing capacity network device, such as a router. It adopts an online, unsupervised intrusion detection approach based on an ensemble of Autoencoders that consist of training a set of Autoencoders on clustered statistical features of a defined size. The authors show that Kitsune performs better than other ML algorithms, such as Gaussian Mixture Models (GMM) and PcStream2. Nonetheless, designing Kitsune to be light weight limits Autoencoder size and the exploration of more complex architectures. Furthermore, the authors consider aggregated features for different time-windows without detailing their choice of the time-windows.

### B. Novelty of our work in comparison to the literature

Motivated by the above works, we first analyze the impact of flow-based features on the detection of multiple DDoS attacks using an Autoencoder. We show that a flow-based Autoencoder fails to provide satisfactory detection performance for TCP SYN, UDPLag, NetBIOS, and PortMap attacks. Therefore, we explore the impact of time-based features on detecting these attacks. Unlike [6] that considers an ensemble of three-layer Autoencoders, each trained on a subset of features over aggregated time-windows, we evaluate the performance of a time-based Autoencoder in detecting anomalies in a non-constrained environment. We employ more complex Autoencoders to support correlation between a larger set of features.

We perform a fine-grained analysis on the impact of different time-windows on the detection of multiple DDoS attacks. In contrast, Kitsune only shows the impact of aggregated time-windows. Based on the results of each time-window on the detection rate, we choose and evaluate the performance of aggregated time-windows. Furthermore, we study the impact of threshold selection for the reconstruction error provided by the Autoencoder. We design a heuristic for threshold selection that maximizes the F1 score for single and aggregated time-based feature enabled Autoencoders. Finally, we showcase that our proposed time-based anomaly detection approach provides high detection rate in comparison to its flow-based counterpart.

## III. Time-based anomaly detection system

### A. Overview

We develop an anomaly detection system that considers monitoring network traffic and detecting any deviation from the normal behavior as an anomaly. The detection is done through a neural network, primarily an Autoencoder. Our system accounts for training and execution modes. During the training mode, the Autoencoder learns normal network behavior. When in execution mode, the trained Autoencoder applies its learning to detect anomalies. To better explain our time-based anomaly detection system, Fig. 1 presents a flow diagram detailing its different building blocks.

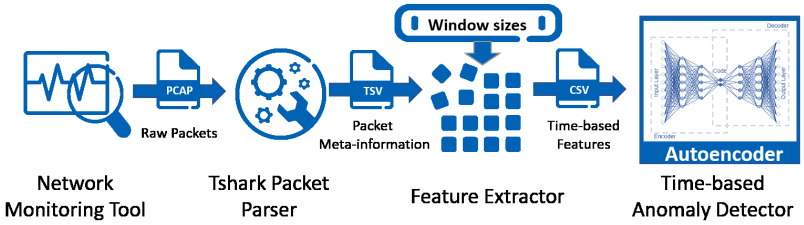


Fig. 1. Time-based anomaly detection system

The presented system is composed of a network monitoring tool, such as Wireshark [23], which monitors network traffic and collects raw packet data during normal network operation. When in training mode, the collected packets represent benign data summarized in a PCAP file (*cf.*, Fig. 1). The PCAP file is then parsed by TShark [24], a network protocol analyzer. The TShark packet parser receives raw binary, parses the packets and extracts meta information (*e.g.*, source/destination IP, port numbers, frame length, *etc.*) into a TSV file. The meta information is fed to a feature extractor [6], to extract time-based features, based on a set of provided time-windows. The extracted features are generated in a CSV file and provided as input to the Autoencoder. Hence, the Autoencoder is trained on benign data expressed by the provided time-based features.

Once trained, the Autoencoder can be used to detect anomalies during the operation of the network. Similar to the training phase, raw packets are collected, parsed and pre-processed to extract time-based features, while considering the same time-windows used during the training phase. During the execution phase, the trained Autoencoder provides a low reconstruction error when received data is similar to what it has been trained on (*i.e.*, benign data). However, the Autoencoder will fail to accurately reconstruct anomalous data that it has not been trained on. This will lead to a high reconstruction error indicating the detection of an anomaly. More details on feature extraction and the Autoencoder will be provided in the following subsections.

### B. Feature selection and extraction

As our anomaly detection system is highly dependant on learning network traffic patterns, it is crucial to represent these patterns accurately to enhance the detection accuracy of our system. This translates into selecting and extracting a set of features that depict the observed traffic [6]. Many schemes have been used to represent network traffic, including:

- *Flow-based features* represent statistical values that describe the set of packets within a flow. Examples of such features include, but are not limited to, packet count, average packet size, inter-packet arrival times, *etc*. CICFlowMeter is one of the well-known tools for generating flow-based features. It generates 84 features per flow.
- *Packet-based features* are more fine-grained than flow-based features as they describe each packet in the network. Examples of such features include, but are not limited to, packet size, source IP, destination IP, *etc*.

Flow-based features have been widely used in the literature, as discussed in Section II. However, they fail to assist ML models in providing satisfactory anomaly detection rate for some DDoS attacks (*e.g.*, TCP SYN, UDPLag, NetBIOS), as shown in Section V. Packet-based features describe each packet. However, they fail to capture the context and the packet's relationship with other packets in the network.

For this purpose, the authors of Kitsune [6] discuss the importance of temporal statistical features in detecting anomalies. They explain that a sudden increase in jitter may indicate that the traffic, which seems legitimate, is generated by a man-in-the-middle attack. Such increase can not be reflected by flow-based features nor non-temporal ones. Hence, the authors develop a feature extraction tool for statistical time-based features from packets exchanged during a time-window. A time-window is defined as a specific time period.

For each arriving packet, Kitsune's feature extractor generates 20 traffic statistics for the input time-windows. For instance, considering a time-window of 10 ms, for a packet $P$ captured at time $t$, Kitsune's feature extractor first constructs the following sets using the packets captured from time $t$ to $t + 10$:

- All packets originating from the same IP and MAC of $P$
- All packets originating from the same IP of $P$
- All packets with the same source and destination IPs of $P$
- All packets with the same source and destination sockets of $P$

For each of these aggregation levels, the feature extractor computes a few statistics, such as mean, standard error, and correlations of bandwidth of the outbound traffic, bandwidth of the outbound and inbound traffic together, packet rate of the outbound traffic, and inter-packet delays of the outbound traffic. Having the features for packet $P$, the feature extractor considers the next captured packet and repeats the above steps to generate the corresponding features for that packet. The feature extractor of Kitsune uses incremental statistics maintained over a damped window *i.e.*, an incremental statistic can be deleted when its dampening weight becomes zero, to save additional memory [6].

We adopt Kitsune's feature extractor to evaluate the impact of different time-windows along with aggregated time-based features (*i.e.*, features from multiple time-windows *e.g.*, 10sec and 1sec, denoted $w$=10sec,1sec), to detect anomalies corresponding to DDoS attacks. It is worth mentioning that time-based features can be generated under different network conditions in an online or offline scenario either, using PCAP

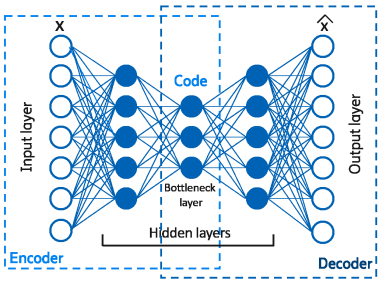


Fig. 2. Autoencoder with 3 hidden layers and 7 input/output neurons

files (as in this work) or through network probes. They represent statistical information over a subset of packets, similar to flow-based features that account for all the packets in a flow [25], [26]. In Section V, we showcase the efficacy of temporal features in detecting anomalies for TCP SYN, UDPLag, NetBIOS and PortMap DDoS attacks, which are hardly detected by flow-based features.

### C. Autoencoder for anomaly detection

We leverage the extracted time-based features to train a neural network, primarily an Autoencoder, to learn the benign network traffic behavior.

#### 1) Overview

As briefly discussed in Section I, an Autoencoder tries to map the input data to a lower dimension, such that the resulting lower order representation remains rich enough to reproduce the input data. More specifically, an Autoencoder is composed of different layers [4]:

- *An input layer* with size (*i.e.*, number of neurons) equal to the number of input features.
- *One or multiple hidden layers* of different sizes that encode and decode the input features. Typically, the encoding hidden layers have smaller dimensions than the number of input features. The encoded features are reconstructed in reverse through the decoding hidden layers with dimensions that are typically the reverse of the encoding layers.
- *An output layer* with the same size as the input layer, representing the reconstructed features.

The size and the number of layers of an Autoencoder define its architecture. Fig. 2 represents an Autoencoder with 3 hidden layers and 7 input neurons.

#### 2) Architecture

The number of hidden layers and the number of neurons in each hidden layer, play a crucial role in the performance of an Autoencoder. A deeper Autoencoder may not necessarily be better, as a higher number of parameters could negatively impact convergence time. Furthermore, too many parameters increase network complexity and can introduce high randomness in the learning process, preventing the model from converging to the optimal minimum. On the contrary, too few neurons in the bottleneck layer, may not capture the characteristics of the input features. Hence, it is important to select the appropriate architecture for our time-based anomaly detector that can realize an acceptable convergence time, while providing acceptable detection performance.

#### 3) Reconstruction error and anomaly detection

To better explain the functionality of our time-based anomaly detector, we consider an Autoencoder with a bottleneck layer (*i.e.*, code) of size $z$. The input data of $\mathbf{x} \in R^N$ passes through the encoder part with weights $W_e$ to produce its corresponding mapping $y \in R^z$, which is then used to reconstruct the input data. The reconstructed data is represented by $\hat{\mathbf{x}}$. Having the training data of size $M$, in each training step, the Autoencoder tries to minimize the reconstruction error *i.e.*, the Mean Squared Error (MSE) in our case, between $\mathbf{x}$ and $\hat{\mathbf{x}}$. The MSE is given as:

$$MSE = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(\mathbf{x}_i[j] - \hat{\mathbf{x}}_i[j])^2 \quad (1)$$

where, $N$ is the size of the input features and $\mathbf{x}_i[j]$ represents the $j$th element of sample $\mathbf{x}_i$.

With the reconstruction error loss function, the Autoencoder learns a good lower order mapping that can be used to reconstruct the input data [27]. By observing the samples in the training dataset over numerous epochs, the Autoencoder provides a low MSE when tested using a data similar to the one it was trained on. In contrast, if the test data expresses different behavior from the training data, there is a high chance that the Autoencoder will provide a large MSE between $\mathbf{x}$ and $\hat{\mathbf{x}}$. Thus, considering this behavior, we feed the Autoencoder with benign traffic, to train it to efficiently detect anomalies.

### D. Reconstruction error threshold selection

Recall that our time-based anomaly detector, trained in an unsupervised manner (*i.e.*, using unlabeled data), aims at minimizing the reconstruction error (*i.e.*, MSE). A high value of the reconstruction error depicts a divergence from the normal behavior. To identify such divergence, a threshold must be selected, such that, if the reconstruction error for an input instance is above the threshold, the input is deemed anomalous. However, it is non-trivial to select this threshold. A naïve approach may select the largest training reconstruction error as the threshold. However, the training dataset may contain benign outliers that result in abnormally high reconstruction losses. A very high threshold would result in high false negatives (*i.e.*, low recall) and missing attacks. In contrast, a very low threshold would cause a lot of false alarms, negatively impacting precision. In both cases, the performance of the Autoencoder is significantly degraded. F1-score, ideally 1, takes both false negatives and false positives into consideration. Hence, in our approach, we select the threshold that maximizes the F1-score.

To select the threshold, we reserve a small portion of the test dataset, called optimization dataset (*cf.*, Section IV). Then, we take the benign instances in the optimization dataset and generate their reconstruction errors. Note that there may be outliers with abnormally high reconstruction errors, as before. Therefore, naïvely selecting the highest reconstruction error as the threshold could jeopardize the Autoencoder performance. Instead, we sort the reconstruction errors in descending order,

TABLE I
THE TIMING OF ATTACKS IN THE CICDDOS2019 DATASET

| Type | Attack | Time |
|---|---|---|
| Reflection-based | PortMap | March 11th, 9:43 - 9:51 |
| | LDAP | March 11th, 10:21 - 10:30 |
| | MSSQL | January 12th, 11:36 - 11:45 |
| | NetBIOS | January 12th, 11:50 - 12:00 |
| | SNMP | January 12th, 12:12 - 12:23 |
| | SSDP | January 12th, 12:27 - 12:37 |
| | WebDDoS | January 12th, 13:18 - 13:29 |
| | TFTP | January 12th, 13:35 - 17:15 |
| Exploitation-based | UDP | January 12th, 12:45 - 13:09 |
| | UDPLag | January 12th, 13:11 - 13:15 |
| | SYN | January 12th, 13:29 - 13:34 |

and iterate over the highest $\alpha\%$ of the reconstruction errors in steps of $\alpha/\beta$, selecting the corresponding reconstruction error as the current threshold and predicting on the optimization dataset. At each iteration, we compute the F1-score for the predictions. After exhausting all the thresholds, we select the one that results in the highest F1-score as the optimal threshold for the Autoencoder. The pseudo-code for our threshold selection heuristic is shown in Algorithm 1.

**Algorithm 1** Reconstruction error threshold selection

**Input:** $\alpha$, $\beta$, $model$, $dataset$, $labels$
**Output:** $optimal\_threshold$

$optimal\_threshold \leftarrow optimal\_f1 \leftarrow -\infty$
$mse \leftarrow model.predict(dataset).mse()$
$benign\_mse \leftarrow mse.get\_benign\_mse().sort()$
**for each** $i$ in $range(0, \beta, \alpha/\beta)$ **do**
  $threshold \leftarrow benign\_mse[benign\_mse.len() - i]$
  $predicted\_labels \leftarrow get\_labels(mse, threshold)$
  $f1 \leftarrow get\_f1(labels, predicted\_labels)$
  **if** $optimal\_f1 < f1$ **then**
    $optimal\_threshold \leftarrow threshold$
    $optimal\_f1 \leftarrow f1$
  **end if**
**end for**
**return** $optimal\_threshold$

## IV. DATASET PREPARATION

### A. CICDDoS2019 dataset

We leverage the CICDDoS2019 dataset, which includes 13 different DDoS attacks, carried out via application layer protocols over TCP/UDP. The dataset contains both raw packets of network traffic in PCAP format and flow-based features in CSV format, extracted using CICFlowMeter. The data is captured during two days, on January 12th between 10:30 and 17:15 and on March 13th between 09:40 and 17:35. Multiple reflection- and exploitation-based DDoS attacks are performed at different times, with ones we evaluate shown in Table I.

### B. Data pre-processing

Our time-based anomaly detection Autoencoder is trained on benign traffic and tested on attacks. For this reason, we pre-process the CICDDoS2019 dataset's PCAP files. We first create a training dataset of benign packets, by extracting the data collected between 10:30 and 11:36 on January 12th and disregarding all the attack packets collected during this period. We further add to the created benign dataset, all benign packets collected during different time frames when no attacks were performed (*e.g.*, January 12th from 11:46 till 11:49). There are a total of 243,709 packets in the benign dataset. In order to test the detection performance of our Autoencoder, we construct a separate test dataset for each of the 11 attacks that are shown in Table I. The test files, that we refer to as attack files, contain both benign and attack packets.

TABLE II
TEST DATASET STATISTICS

| Attacks | Benign Packets | Attack Packets |
|---|---|---|
| PortMap | 130249 | 380815 |
| LDAP | 889 | 463928 |
| MSSQL | 2186 | 4997914 |
| NetBIOS | 14291 | 1582576 |
| SNMP | 1910 | 4998090 |
| SSDP | 12993 | 4987006 |
| WebDDoS | 784 | 4999216 |
| TFTP | 13447 | 4986553 |
| UDP | 15390 | 4984610 |
| UDPLag | 3166 | 3123705 |
| SYN | 4863 | 3775195 |

In CICDDoS2019, all attacks originate from a single node. We have used this property to label the packets. Any packet that has the source or destination IP of the attacker is considered an attack packet, while others are benign. We have limited the size of each attack file to 5 million packets. The number of packets considered in each test file are presented in Table II. Furthermore, we randomly extract 1% of the packets from each attack file for the optimization dataset to select the reconstruction error threshold (*cf.*, Section III-D). From Table II, we note an imbalance between the benign and attack packets in the test files. This is primarily because these files are based on the attack time frames *i.e.*, when the attack is performed. Therefore, the number of captured attack packets are larger than the number of benign ones.

## V. EXPERIMENTS

We carry out extensive experiments to evaluate the performance of our time-based anomaly detection system against a flow-based approach. Both solutions leverage an Autoencoder that is trained and tested on the CICDDoS2019 dataset. The presented Autoencoders learn from different features reflecting statistical information about benign flows and packets for the flow-based and time-based anomaly detection, respectively. However, in an operational network, access to exclusively benign data for training the Autoencoder is non-trivial. We assume that the benign data will be much greater than attack data. Hence, without loss of generality, the Autoencoder can be trained on a mix of unlabeled benign and attack (*i.e.*, in much smaller proportion) traffic. The Autoencoder must be re-trained whenever a performance degradation is noticed (*e.g.*, uncovering a high number of false positives). Note that identified anomalies must be further investigated by either a security expert, or other methods that can identify the attack

type or confirm the anomaly.

### A. Environmental setup

We perform our evaluation on a virtual machine deployed in an OpenStack [28] environment. The virtual machine runs Ubuntu 18.04 and is managed by a KVM hypervisor. It includes 4 vCPUs, running on top of a server featuring 4 x 64 bit intel core processor CPUs. The virtual machine also uses a GPU and features 28GB of RAM. Our anomaly detection is implemented using Python 3.7.6. The presented Autoencoders leverage the Keras library of Tensorflow 2.2.0 [29].

### B. Evaluation metrics

We use the Autoencoder as a binary classifier to predict input instances (*i.e.*, flows or packets) belonging to the benign or the attack class. Accuracy is a widely used metric to evaluate classification performance. It is the proportion of true predictions among the total number of predictions. However, it suffers in the face of dataset imbalance. For example, consider a classification problem with 95% and 5% instances belonging to the attack and benign classes, respectively. Even if the classifier predicts all input instances as attack, the accuracy would still be 95%, which is misleading. Therefore, we rely on the following metrics to evaluate the prediction performance of the Autoencoder.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3}$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

We define the attacks as positive instances. Precision is the proportion of attack predictions corresponding to the attack class. In contrast, recall is the proportion of correct predictions for the attack class. A higher precision suggests lower false alarms *i.e.*, benign instances being predicted as attack, while a high recall implies that more attack instances are not missed. The F1-score is a harmonic mean of precision and recall, depicting the trade-off between these metrics. Our primary objective is to maximize the F1-score.

### C. Anomaly detection using flow-based features

The evaluation of the Autoencoder with flow-based features is performed on Kaggle [30], using features from CICFlowMeter. These flow features are provided in CSV files as part of the dataset. We exclude features, such as source and destination IP addresses and ports, as they can introduce a bias in model behavior. We employ Autoencoders with different number of hidden layers and neurons per layer, and tune various hyper-parameters to detect anomalies. We achieve reasonable performance in anomaly detection, with F1-scores greater than 90% using flow-based features, in the face of most attacks in the CICDDoS2019 dataset.

However, the Autoencoder is unable to accurately model the benign behavior using flow-based features. Therefore, for NetBIOS, PortMap, TCP SYN and UDPLag attacks, the Autoencoder shows a lackluster performance. For example, consider the SYN attack that abuses the TCP three-way handshake procedure, and floods the server with repetitive SYN packets. Though the flow-based features employed in our evaluation include the flow's SYN flag count, among other flow statistics, the Autoencoder performs poorly in detecting anomalies pertaining to the SYN flooding attack.

Clearly, the flow-based features are not discriminative enough to distinguish between benign and anomalous behavior. Table III highlights the performance of the Autoencoder with flow-based features. We only show the attacks that result in poor performance. These results are generated using $[79, 50, 25, 10, 25, 50, 79]$ as the Autoencoder architecture with a batch size of 32. ReLU and Linear activation functions are used for the hidden layers and the output layer, respectively. Note that the choice of the Autoencoder architecture does not have a significant impact on anomaly detection performance.

TABLE III
ANOMALY DETECTION USING FLOW-BASED FEATURES

| Attack | Precision | Recall | F1-score |
|---|---|---|---|
| NetBIOS | 0.54794 | 0.09378 | 0.16016 |
| PortMap | 0.43317 | 0.07944 | 0.13426 |
| SYN | 0.43803 | 0.10201 | 0.16548 |
| UDPLag | 0.59978 | 0.15091 | 0.24115 |

### D. Anomaly detection using time-based features

#### 1) Autoencoder selection

Before we experiment with the time-based features to detect anomalies, we select the appropriate Autoencoder architecture and hyper-parameters. We evaluate four different architectures, starting from a very shallow network (*i.e.*, one hidden layer), and increase it to seven hidden layers. Consider $k$ = number of input and output neurons, the architectures are: (i) $Arch.\ A = [k, k \times 30\%, k]$, $Arch.\ B = [k, k \times 70\%, k \times 30\%, k \times 70\%, k]$, $Arch.\ C = [k, k \times 80\%, k \times 50\%, k \times 30\%, k \times 50\%, k \times 80\%, k]$, and $Arch.\ D = [k, k \times 80\%, k \times 60\%, k \times 40\%, k \times 30\%, k \times 40\%, k \times 60\%, k \times 80\%, k]$. Note that we do not hard-code the number of neurons in the hidden layers. Rather, we specify them as a percentage of the neurons in the input and output layers. For example, in $Arch.\ A$, there is one hidden layer, with the number of neurons equal to 30% of the number of input neurons *i.e.*, for $k = 20$, the number of neurons in the singleton hidden layer is 6. We leverage $k = 80$ by aggregating time-based features from different window sizes, denoted $w$ (*cf.*, Section V-D3), including 10sec, 1sec, 100ms, and 10ms, and observe the average reconstruction loss *i.e.*, the average MSE across the different training epochs. The Autoencoder hyper-parameters are depicted in Table IV. Note that during our experiments, the batch sizes affect training time, without significant impact on anomaly detection performance. Given the space limitation, we do not report on corresponding results.

All architectures, except $Arch.\ D$ depict reasonable convergence within the first 50 epochs, as shown in Fig. 3. In contrast, $Arch.\ D$ takes the longest to converge, and continues to reduce the average MSE upto 200 epochs. Indeed, it is pos-

TABLE IV
AUTOENCODER SETTINGS

| Hyper-parameter | Value |
|---|---|
| Number of epochs | 200 |
| Patience | 20 |
| Learning rate | 0.001 |
| Batch size | 1024 |
| Validation split | 0.2 |
| Optimizer | Adam |
| Hidden activation | ReLU |
| Output activation | Linear |

sible for $Arch.\ D$ to converge further to smaller average MSE. However, this comes with the largest number of parameters, with no guarantee on convergence. $Arch.\ A$ is the leanest architecture with the smallest degree of freedom, limiting its performance. On the other hand, $Arch.\ B$ and $Arch.\ C$ show comparable performance. However, $Arch.\ C$, with its higher degree of freedom, further minimizes the average MSE over the training epochs, resulting in the smallest average MSE across all architectures. Hence, we choose $Arch.\ C$ for the remaining of our experiments.

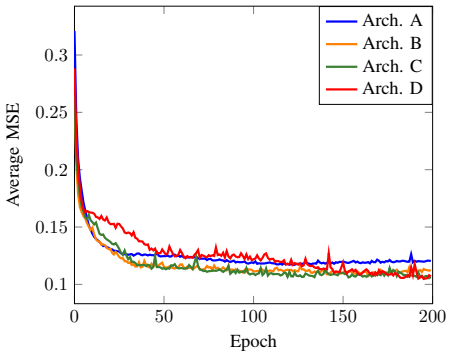


Fig. 3. Average MSE over training epochs with $w$ = 10sec,1sec,100ms,10ms

*2) Reconstruction error threshold selection*

Recall that the Autoencoder is primarily trained on benign data in an unsupervised manner (*i.e.*, using unlabeled data), to minimize the reconstruction loss (*i.e.*, MSE). After training, the Autoencoder flags any divergence from the norm as an anomaly. We randomly extract 1% of the packets from each attack file for the optimization dataset, while preserving the same proportion of attack and benign packets existing in that file. We observe that the outliers in the benign portion of the optimization dataset are minimal. Hence, the optimal threshold is typically amongst the first few reconstruction errors, as depicted in Fig. 4 (*i.e.*, the first in this case, highlighted in yellow). We employ $\alpha = 20\%$ and $\beta = 20$ in our selection of the threshold. Increasing $\beta$ (*i.e.*, the granularity) allows for further exploration, without significant impact on Autoencoder performance (*cf.*, Section V-D3). Nevertheless, both $\alpha$ and $\beta$ can be adjusted depending on optimization dataset quality.

*3) Impact of window sizes*

In the previous subsections, we use aggregated window sizes (*i.e.*, $w$=10sec,1sec,100ms,10ms) to evaluate the impact of Autoencoder architectures on training performance and showcase

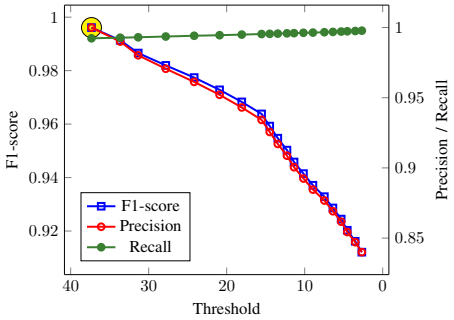


Fig. 4. Maximizing F1-score during threshold selection with $\alpha = 20\%$, $\beta = 20$, and $w$ = 10sec,1sec,100ms,10ms, and the optimal F1-score highlighted

our threshold selection heuristic. The CICDDoS2019 dataset has traces for numerous reflection- and exploitation-based attacks, as shown in Table I. Indeed, these attacks, within and across categories, can evolve differently. For example, a flooding attack can evolve quickly over time. Hence, a smaller time-window can potentially flag corresponding packets as anomalous, facilitating early detection. In contrast, a slow and low attack will be more stealthy with malicious activity spanning across a larger time-window. In this subsection, we start by evaluating the impact of different window sizes (*i.e.*, $w$=10sec, $w$=1sec, $w$=100ms, and $w$=10ms) on anomaly detection, with respect to the different attacks.

Time-based features facilitate the Autoencoder in capturing the true characteristics of the benign traffic. This is also supported by the lower average reconstruction loss, as depicted in Fig. 3. As shown in Table V, the Autoencoder is able to achieve significantly higher F1-scores for the NetBIOS, PortMap, SYN and UDPLag attacks, in comparison to Table III. The Autoencoder with time-based features is less sensitive to the remaining attacks, and performs equally well in comparison to its flow-based counterpart. However, the window size impacts the performance of the Autoencoder for some attacks (examples are highlighted in yellow in Table V). Evidently, Autoencoder performance in anomaly detection for LDAP and PortMap decreases as the window size is reduced, from $w$=10sec through $w$=10ms. In contrast, for NetBIOS, the Autoencoder performance increases as the window size is reduced, from $w$=10sec through $w$=10ms. Clearly, the Autoencoder is susceptible to window sizes, primarily in the face of reflection-based attacks.

Choosing a window size that allows to detect anomalies pertaining to the various attacks, specifically unknown attacks, is non-trivial. We show that aggregating time-based features across multiple window sizes achieves superior performance in anomaly detection across all attacks in the CICDDoS2019 dataset. Note that the higher the number of aggregated window sizes, the more complex the Autoencoder with a higher number of parameters to tune for convergence. Therefore, although aggregating time-based features across a large number of window sizes may seem plausible, it is not a recommended solution. We experiment with two aggregation levels *i.e.*, $w$=10sec, 1sec, 100ms, 10ms, and $w$=10sec, 10ms, respectively. As shown in Fig. 5, the smaller aggregation across only

TABLE V
ANOMALY DETECTION USING TIME-BASED FEATURES AND INDIVIDUAL WINDOW SIZES OF 10SEC, 1SEC, 100MS AND 10MS

| | $w$ = 10sec | | | $w$ = 1sec | | | $w$ = 100ms | | | $w$ = 10ms | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Attack | Precision | Recall | F1-score | Precision | Recall | F1-score | Precision | Recall | F1-score | Precision | Recall | F1-score |
| LDAP | 1.00000 | 0.99749 | 0.99875 | 1.00000 | 0.98747 | 0.99369 | 1.00000 | 0.93985 | 0.96899 | 1.00000 | 0.90100 | 0.94792 |
| MSSQL | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.99840 | 0.99920 | 1.00000 | 0.99467 | 0.99733 | 1.00000 | 0.99360 | 0.99679 |
| NetBIOS | 1.00000 | 0.36730 | 0.53727 | 1.00000 | 0.40628 | 0.57781 | 1.00000 | 0.83115 | 0.90779 | 1.00000 | 0.97425 | 0.98696 |
| PortMap | 0.99988 | 0.97332 | 0.98642 | 0.99984 | 0.95073 | 0.97466 | 0.99997 | 0.90411 | 0.94963 | 0.99994 | 0.85394 | 0.92119 |
| SNMP | 1.00000 | 0.99942 | 0.99971 | 1.00000 | 0.99884 | 0.99942 | 1.00000 | 0.99476 | 0.99737 | 1.00000 | 0.99126 | 0.99561 |
| SSDP | 1.00000 | 0.99791 | 0.99895 | 1.00000 | 0.99693 | 0.99846 | 1.00000 | 0.99104 | 0.99550 | 1.00000 | 0.99221 | 0.99609 |
| SYN | 1.00000 | 0.99954 | 0.99977 | 1.00000 | 0.99886 | 0.99943 | 1.00000 | 0.99520 | 0.99759 | 1.00000 | 0.98263 | 0.99124 |
| TFTP | 1.00000 | 0.99058 | 0.99527 | 1.00000 | 0.99083 | 0.99539 | 1.00000 | 0.98694 | 0.99343 | 1.00000 | 0.97066 | 0.98511 |
| UDP | 1.00000 | 0.99827 | 0.99913 | 1.00000 | 0.99581 | 0.99790 | 1.00000 | 0.98960 | 0.99477 | 1.00000 | 0.98650 | 0.99320 |
| UDPLag | 1.00000 | 0.99965 | 0.99982 | 1.00000 | 0.99860 | 0.99930 | 1.00000 | 0.99227 | 0.99612 | 1.00000 | 0.98244 | 0.99114 |
| WebDDoS | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 0.99574 | 0.99786 | 1.00000 | 0.98864 | 0.99429 |

two window sizes (*i.e.*, $w$=10sec, 10ms) achieves anomaly detection F1-score over 99% for most attacks and greater than 95% for all attacks, while outperforming aggregation across all window sizes. The training time for this aggregation level, with GPU acceleration, is 798 seconds.

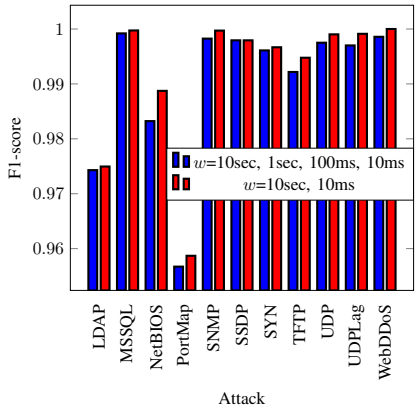


Fig. 5. Time-based anomaly detection using aggregated window sizes

### 4) Robustness to zero-day attacks

Sophisticated adversaries often randomize malicious activities or leverage polymorphic attacks to avoid detection. Therefore, anomaly detection should be robust against unknown attacks or variations of known attacks. To evaluate the robustness of the Autoencoder, we modify threshold selection, such that, we exclude some attacks (*i.e.*, NetBIOS, PortMap, and SYN) from the optimization dataset. Hence, during threshold selection, the reconstruction errors depicting the characteristics of these *unknown* attacks do not influence the selection of the optimal threshold. As shown in Table VI, anomaly detection in the face of NetBIOS and TCP SYN attacks achieve an F1-score of over 99%. The performance against PortMap is just over 83%, which is primarily attributed to a high number of false alarms. Nevertheless, the Autoencoder with time-based features, significantly outperforms its flow-based counterpart in anomaly detection, even in the face of unknown attacks.

TABLE VI
ROBUSTNESS TO UNKNOWN ATTACKS USING TIME-BASED FEATURES AND AGGREGATED WINDOW SIZES OF 10SEC AND 10MS

| | $w$ = 10sec, 10ms | | |
|---|---|---|---|
| Attack | Precision | Recall | F1-score |
| LDAP | 1.00000 | 0.99875 | 0.99937 |
| MSSQL | 1.00000 | 1.00000 | 1.00000 |
| NetBIOS | 1.00000 | 0.99603 | 0.99801 |
| PortMap | 0.73421 | 0.97541 | 0.83780 |
| SNMP | 1.00000 | 1.00000 | 1.00000 |
| SSDP | 1.00000 | 0.99935 | 0.99967 |
| SYN | 1.00000 | 0.99959 | 0.99979 |
| TFTP | 0.99967 | 0.99802 | 0.99884 |
| UDP | 1.00000 | 0.99913 | 0.99957 |
| UDPLag | 1.00000 | 0.99965 | 0.99982 |
| WebDDoS | 1.00000 | 1.00000 | 1.00000 |

## VI. CONCLUSION

We propose a novel time-based anomaly detection system, that leverages an Autoencoder to detect anomalous DDoS traffic. We evaluate the impact of different window sizes in detecting anomalies pertaining to different DDoS attacks, and show that aggregating features across just two time-windows can achieve a detection F1-score of over 99% for most attacks and greater than 95% for all attacks. We also show the robustness of our approach to unknown attacks with an F1-score of over 99% for TCP SYN and NetBIOS, and just over 83% for PortMap. In the future, we will explore other window sizes and their impact on PortMap, to harden Autoencoder robustness to unknown or polymorphic attacks. We will also investigate the performance of our time-based Autoencoder using more sophisticated threshold selection techniques, and compare with state-of-the-art approaches.


## ACKNOWLEDGMENTS

We would like to thank Dr. Makan Pourzandi, Dr. Stere Preda, Dr. Parisa Heidari, Richard Brunner, and Adel Larabi from Ericsson Canada, for their invaluable feedback. We would also like to thank Lechuan Peng for his initial work on the Autoencoder. This work is supported in part by Ericsson Canada, and in part by the NSERC CRD Grant CRDPJ 536445-18.

## REFERENCES


[1] H. S. Lallie, L. A. Shepherd, J. R. Nurse, A. Erola, G. Epiphaniou, C. Maple, and X. Bellekens, "Cyber security in the age of covid-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic," *arXiv preprint arXiv:2006.11929*, 2020.

[2] O. Kupreev, E. Badovskaya, and A. Gutnikov, "Ddos attacks in q1 2020," 2020. [Online]. Available: https://securelist.com/ddos-attacks-in-q1-2020/96837/

[3] S. Sedaghat, "The forensics of ddos attacks in the fifth generation mobile networks based on software-defined networks." *IJ Network Security*, vol. 22, no. 1, pp. 41–53, 2020.

[4] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Ddosnet: A deep-learning model for detecting network attacks," *21 ST IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WOWMOM 2020), CCNCPS2020 Workshop*, 2020.

[5] J. Lam and R. Abbas, "Machine learning based anomaly detection for 5g networks," *arXiv preprint arXiv:2003.03474*, 2020.

[6] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," *Network and Distributed System Security Symposium (NDSS 2018)*, no. February, 2018.

[7] F. Erlacher and F. Dressler, "On high-speed flow-based intrusion detection using snort-compatible signatures," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[8] M. Masdari and H. Khezri, "A survey and taxonomy of the fuzzy signature-based intrusion detection systems," *Applied Soft Computing*, p. 106301, 2020.

[9] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[10] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[12] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, 2018.

[13] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 29–35.

[14] S. Sarraf *et al.*, "Analysis and detection of ddos attacks using machine learning techniques," *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, vol. 66, no. 1, pp. 95–104, 2020.

[15] M. Kravchik and A. Shabtai, "Efficient Cyber Attacks Detection in Industrial Control Systems Using Lightweight Neural Networks and PCA," pp. 1–18, 2019. [Online]. Available: http://arxiv.org/abs/1907.01216

[16] K. Yang, J. Zhang, Y. Xu, and J. Chao, "Ddos attacks detection with autoencoder," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[17] H. Choi, M. Kim, G. Lee, and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *Journal of Supercomputing*, vol. 75, no. 9, pp. 5597–5621, 2019. [Online]. Available: https://doi.org/10.1007/s11227-019-02805-w

[18] Y. Intrator, G. Katz, and A. Shabtai, "MDGAN: boosting anomaly detection using multi-discriminator generative adversarial networks," *CoRR*, vol. abs/1810.05221, 2018. [Online]. Available: http://arxiv.org/abs/1810.05221

[19] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," *Proceedings - International Carnahan Conference on Security Technology*, vol. 2019-Octob, no. Cic, 2019.

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

[21] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *IEEE communications surveys & tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[22] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers & Security*, vol. 70, pp. 238–254, 2017.

[23] The Wireshark team, "Wireshark, Go Deep," 2020, accessed: 2020-07-24. [Online]. Available: https://www.wireshark.org/

[24] ——, "The Wireshark Network Analyzer," 2020, accessed: 2020-07-24. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[25] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 5–13, 2012.

[26] M. Wullink, G. C. Moura, M. Müller, and C. Hesselman, "Entrada: A high-performance network traffic data streaming warehouse," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 913–918.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[28] OpenStack, "Build the future of Open Infrastructure," 2020, accessed: 2020-07-24. [Online]. Available: https://www.openstack.org/

[29] Google Brain Team, "Tensorflow," 2020, accessed: 2020-07-24. [Online]. Available: https://www.tensorflow.org/

[30] Google LLC, "Kaggle: Your Home for Data Science," 2020, accessed: 2020-07-24. [Online]. Available: https://www.kaggle.com/