This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management

Chronos: DDoS Attack Detection using Time-based Autoencoder

Mohammad A. Salahuddin*, Vahid Pourahmadi*†, Hyame Assem Alameddine*‡, Md. Faizul Bari*§, and Raouf Boutaba*
*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
†Department of Electrical Engineering, Amirkabir University of Technology, Tehran, Iran
‡Ericsson Security Research, Montreal, Quebec, Canada
§Spectrum Software and Consulting, Dhaka, Bangladesh

***Abstract*—Cognitive network management is becoming quintessential to realize autonomic networking. However, the wide spread adoption of the Internet of Things (IoT) devices, increases the risk of cyber attacks. Adversaries can exploit vulnerabilities in IoT devices, which can be harnessed to launch massive Distributed Denial of Service (DDoS) attacks. Therefore, intelligent security mechanisms are needed to harden network security against these threats. In this paper, we propose Chronos, a novel time-based anomaly detection system. The anomaly detector, primarily an Autoencoder, leverages time-based features over multiple time windows to efficiently detect anomalous DDoS traffic. We develop a threshold selection heuristic that maximizes the F1-score across various DDoS attacks. Further, we compare the performance of Chronos against state-of-the-art approaches. We show that Chronos marginally outperforms another time-based system using a less complex anomaly detection pipeline, while out classing flow-based approaches with superior precision. In addition, We showcase the robustness of Chronos in the face of zero-day attacks, noise in training data, and a small number of training packets, asserting its suitability for online deployment.**

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

I. Introduction

Cybercrime is expected to cause $10.5 trillion USD in global losses by 2025 [1]. In fact, cybercriminals are taking advantage of the wide spread adoption of the Internet of Things (IoT) devices. These IoT devices often lack sophisticated security mechanisms, and can be compromised to launch Distributed Denial of Service (DDoS) attacks. A recent study estimated an average of 5,200 attacks per month on IoT devices [2]. Indeed, this necessitates intelligent security mechanisms [3] to protect networks against cyber threats.

DDoS attacks can deplete network resources by increasing the network traffic, and thus, prevent legitimate users from accessing the network. Therefore, detecting DDoS attacks is quintessential to protect the network from severe revenue losses [4], [5]. Anomaly detection is widely used to identify DDoS attacks. It pertains to profiling the normal network traffic, and identifying any deviation from the norm as an anomaly [6]. Therefore, anomaly detection not only identifies known DDoS attacks but also zero-day attacks, which cannot be identified by using an already known attack signature [7]. However, detecting deviations is challenging, as the boundary between anomalous and normal traffic is often non-precise and malicious actions can be tailored to appear benign [6].

Machine Learning (ML) is an ideal tool to establish the normal behavior of a protected network [8], and facilitate cognitive security management. ML techniques are primarily supervised, or unsupervised. Supervised ML techniques [9], [10] provide high precision [11], but require labels that clearly identify all benign and anomalous traffic. As labels are usually hard to obtain [6], [12], unsupervised ML techniques are most typically used for outlier detection [11]–[15]. They overcome the need for labels by assuming that the behavior of the benign traffic is different from that of the anomalous traffic [6], [12], [16]. The properties of the anomalous traffic are then used to detect the deviating (*i.e.*, outlier) class.

Neural networks learn complex, non-linear relationships in data. This facilitates superior performance in comparison to classic ML algorithms, which suffer from high false alarms on large datasets [5], [15]. Autoencoder, a neural network, has been widely adopted for unsupervised anomaly detection. It is composed of: (i) an encoder that compresses the input data into a latent low-dimensional space, and (ii) a decoder that decompresses the compressed data to reconstruct the original input [12], [13]. Hence, the Autoencoder reconstructs the input, while minimizing the reconstruction error. In anomaly detection, the Autoencoder is trained on traffic that is assumed to be benign, at least in majority. This allows the Autoencoder to successfully reconstruct benign traffic with low error, while associating larger reconstruction error with anomalous traffic.

This paper is an extension of our work in [17]. We leverage an Autoencoder to develop Chronos. Chronos is a novel time-based anomaly detection system, which is trained on aggregated time-based features that are extracted from packets observed over varying time-windows. We extensively evaluate the impact of time-windows in characterizing different DDoS attacks by leveraging time-based features. We show that time-based features outperform flow-based statistical features that have been extensively used in the literature for anomaly detection, and discuss the shortcomings of the latter. We perform extensive evaluations on the CICDDoS2019 dataset [18]. Our main contributions are:

- We develop Chronos, a novel time-based anomaly detection system that leverages an Autoencoder, trained using time-

based features, for DDoS attack detection. The time-based features depict statistical information for a subset of packets collected over a specific period or time-window.

- We propose a threshold selection heuristic that maximizes F1-score. Though a threshold is required to differentiate benign and anomalous traffic, it may undermine Autoencoder performance for some DDoS attacks, *i.e.*, the selected threshold may not be optimal for all DDoS attacks. Therefore, we use the Receiver Operating Characteristic (ROC) curves to highlight the efficacy of Chronos, independent of a specific threshold.
- We explore the impact of multiple time-windows and their aggregations on the performance of detecting anomalous DDoS traffic. We show that a single time-window is insufficient to capture the holistic behavior of benign traffic. In contrast, aggregating across only two window sizes result in superior Autoencoder performance, marginally outperforming the state-of-the-art in [15] using a rather simple, less complex anomaly detection pipeline.
- We showcase the robustness of Chronos to zero-day attacks, which we primarily attribute to the time-based features. We reason the slightly inferior performance of PortMap against other attacks by visualizing a lower order representation of the time-based features for PortMap, which shows a significant overlap with benign packets. This is in stark contrast to other DDoS attacks.
- We evaluate the sensitivity of the Autoencoder to noisy training data, *i.e.*, benign training packets contaminated with attack packets. Indeed, as the noise increases, the Autoencoder performance deteriorates. We attribute this to model overfitting, and leverage regularization techniques to alleviate the impact of noise in training data. We also show the effectiveness of the Autoencoder on partial training data (*i.e.*, a small number of packets) with corresponding time-based features, making it suitable for deployment in an online setting with limited data.
- We compare the time-based features against state-of-the art flow-based features. We leverage flow-based features from CICFlowMeter [19] and choose the feature subsets that are inspired from [5] and [20]. Regardless of the flow-based feature subset, the Autoencoder performance is impaired for most attacks, with a lackluster precision. This undermines the suitability of the Autoencoder using flow-based features to detect anomalies with respect to various DDoS attacks.

The rest of the paper is organized as follows. Section II provides a literature review and discusses the novelty of our work in comparison to the state-of-the-art. In Section III, we expose Chronos and its building blocks. The CICDDoS2019 dataset and data pre-processing is discussed in Section IV. Our extensive experimental results and analysis are presented in Section V. In Section VI, we conclude with a brief summary and instigate future research direction.

II. Literature Review

Numerous works in the literature have explored ML techniques for anomaly detection. In the following, we review the closely related works and highlight the novelty of our contributions.

A. Machine learning for intrusion detection

Doshi *et al.* [9] study DDoS attack detection in an IoT environment. They evaluate 5 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 employ stateless features that are derived from flow characteristics of individual packets, along with stateful statistical features collected over a time-window of 10 seconds. Their work is limited to only detecting three types of DDoS attacks, *i.e.*, TCP SYN, UDP flood and HTTP GET flood attacks.

Sarraf *et al.* [10] use the CICIDS2017 dataset to train a DT and a L-SVM for DDoS attack detection. Sharafaldin *et al.* [18] generate the CICDDoS2019 dataset that encompasses 13 different DDoS attacks. They evaluate the performance of 4 classic ML techniques, including Iterative Dichotomiser 3 (ID3), RF, Naïve Bayes, and logistic regression using flow-based statistical features that are extracted using CICFlowMeter. However, their results show poor detection performance with respect to F1-score (*cf.*, Section V-B).

Using the aforementioned CICDDoS2019 dataset and flow-based features, Elsayed *et al.* [5] develop an intrusion detection system against DDoS attacks in an SDN environment, which is based on Recurrent Neural Network (RNN) with an autoencoder. Jia *et al.* [20] propose FlowGuard for the detection, identification and mitigation of IoT DDoS attacks. The authors employ a Long-Short Term Memory (LSTM) model for DDoS attack detection using flow-based features. They also develop a Convolutional Neural Network (CNN) for DDoS attack classification. All the above works consider supervised learning techniques, which require labeled network traffic that is difficult to obtain [6], [12].

As the number of available labeled samples is usually small, semi-supervised learning has been employed. Idhammad *et al.* [21] develop a sequential semi-supervised ML approach for DDoS attack detection. The authors leverage unsupervised and time sliding window approach for detecting anomalous traffic using co-clustering algorithm, entropy estimation and information gain ratio. They use supervised ensemble ML classifiers with the Extra-Tree algorithm for classification of anomalous traffic. Gao *et al.* [22] present a fuzziness-based semi-supervised learning approach via ensemble learning for network intrusion detection. The authors combine a fuzziness-based method to mine the hidden structure of unlabeled data, while leveraging a supervised learning approach via an ensemble of the Classification And Regression Tree (CART) to classify labeled data. Similar to supervised learning, semi-supervised learning depends on the availability of a partially labeled dataset, which is difficult to obtain.

To overcome the supervised and semi-supervised learning shortcomings, unsupervised learning has been employed. Choi *et al.* [13] 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. Yang *et al.* [12] develop an Autoencoder with using sub-flow features to reduce the response time in detecting DDoS attacks. Their Autoencoder performs better than other classic ML algorithms. Villalobos *et al.* [23] present a distributed and collaborative architecture for online high rate DDoS attack detection and mitigation. Their architecture leverages an in-memory distributed graph data structure along with unsupervised ML algorithms, such as K-Means for DDoS attack detection. However, most works that leverage unsupervised learning employ flow-based features, which can only detect a subset of network attacks as they are oblivious to packet-level information [24]–[26].

Mirsky *et al.* [15] 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 across different time-windows, without detailing their choice of the time-windows in the aggregation.

Other works [27]–[30] that consider the notion of time for anomaly detection address the problem using univariate and multivariate time-series. These works aim to detect anomalies in future time-steps based on previous observations, using numerous ML algorithms, such as LSTM [27], variational Autoencoder [28], RNN [29], graph neural networks [30], among others. The authors do not leverage or evaluate the impact of time-based features that are generated by aggregating statistics across different time-windows for detecting various anomalous DDoS traffic.

B. Novelty of our work

Motivated by the above works, we analyze the impact of flow-based features on detecting multiple DDoS attacks using the Autoencoder. We show that a flow-based Autoencoder fails to provide satisfactory detection performance for all DDoS attacks in the CICDDoS2019 dataset. Using flow-based features inspired from [5], [20] results in a high number of false alarms across all DDoS attacks, undermining the suitability of flow-based features for anomaly detection. Therefore, we explore the impact of time-based features, similar to those used in Kitsune [15], for detecting DDoS attacks. We perform a fine-grained analysis on the impact of individual time-windows, while Kitsune only shows the impact of a fixed time-window aggregation across five time-windows. Based on the Autoencoder performance for individual time-windows, we choose and evaluate the performance of different time-window aggregations. We show that aggregating across a small number (*i.e.*, only two) time-windows and a simpler anomaly detection pipeline, Chronos marginally outperforms the state-of-the-art. Furthermore, the time-based features significantly outperform the flow-based features in anomaly detection across all attacks.

Kitsune [15] considers an ensemble approach, named KitNet, which consists of an ensemble of Autoencoders, each trained on a subset of features over aggregated time-windows. Unlike KitNet that is designed for detecting anomalies in a constrained environment (*e.g.*, IoT gateways), we evaluate the performance of Chronos using a time-based, complex Autoencoder that supports correlation among a larger set of features in an unconstrained environment (*e.g.*, network edge servers). Nevertheless, we compare the performance of Chronos against KitNet using the CICDDoS2019 dataset. In fact, we investigate the performance of KitNet on new types of DDoS attacks. We show that Chronos performs slightly better than KitNet across various DDoS attacks. We also discuss the pros and cons of both approaches.

III. Chronos: Time-based Anomaly Detection System

A. Overview

We develop Chronos, a time-based anomaly detection system that monitors network traffic and detects any deviation from the normal behavior as an anomaly. The detection is achieved via a neural network, primarily an Autoencoder. Chronos 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 Chronos, our time-based anomaly detection system, Fig. 1 presents a flow diagram detailing its different building blocks.

The presented system is composed of a network monitoring tool, such as Wireshark [31], 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 [32], 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 [15], to extract time-based features, based on a set of provided time-windows. The extracted features are stored 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 network operation. 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 goal of the trained Autoencoder is to provide a low reconstruction error when the received data is similar to what it has been trained on (*i.e.*, benign data). Hence, we expect the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management

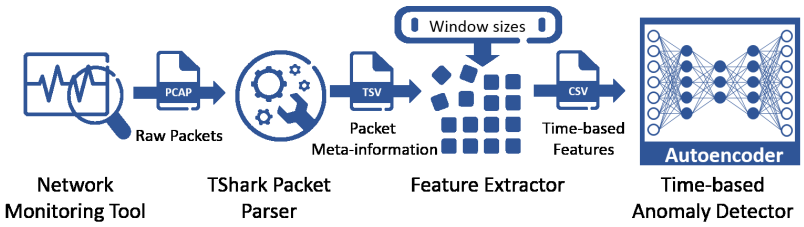


Fig. 1: Chronos: Time-based anomaly detection system

Autoencoder to fail in accurately reconstructing the unseen anomalous data, leading to a high reconstruction error and detection of anomalies. However, it is important to note that the Autoencoder may not always perform well in the face of anomalous data, especially when the training data is noisy [33], which we showcase in Section V-C7 and alleviate its impact. More details on feature extraction and the Autoencoder are provided in the following subsections.

B. Feature selection and extraction

As Chronos is highly dependant on learning network traffic patterns, it is crucial to represent these patterns accurately and enhance the detection accuracy of our system. This translates into selecting and extracting a set of features that depict the observed traffic [15]. 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 85 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 facilitate the ML models in achieving satisfactory anomaly detection performance for numerous DDoS attacks, as shown in Section V. Packet-based features describe each packet. However, they fail to capture the context and a packet's relationship with other packets in the network.

Therefore, the authors of Kitsune [15] 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 an increase can not be reflected by flow-based nor non-temporal features. 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 each of the provided time-window. For instance, considering a time-window of 10 ms, for a packet P captured

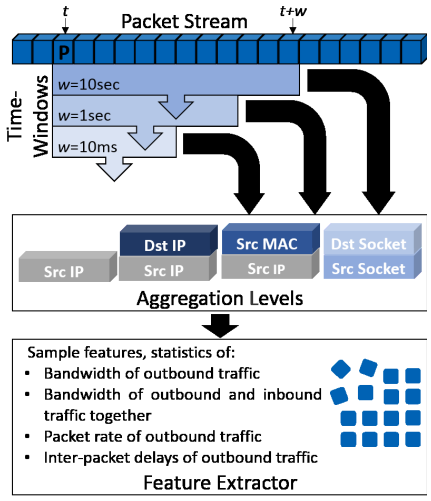


Fig. 2: Time-windows and feature extraction over different packet aggregation levels

at time t, Kitsune's feature extractor first constructs the following sets using the packets captured from time t to $t + 10$, as shown in Fig. 2:

- 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 [15].

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management

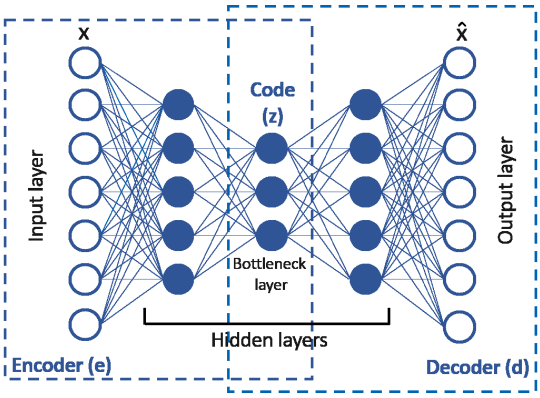


Fig. 3: Autoencoder with 3 hidden layers and 7 input/output neurons

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 using PCAP 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 [34], [35]. In Section V, we showcase the efficacy of temporal features in detecting anomalies for various DDoS attacks, which result in high false alarms using 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 [5]:

- *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. 3 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 in Fig. 3) of size z. The input data $\mathbf{x} \in \mathbb{R}^N$, where $\mathbb{R}^N$ is a N dimensional set of real numbers, passes through the encoder with parameters W_e to produce the corresponding mapping $y \in \mathbb{R}^z$, which is used by the decoder with parameters W_d to reconstruct the input data. The reconstructed data (*i.e.*, output) is represented by $\hat{\mathbf{x}} \in \mathbb{R}^N$. Having training data of size M, in each training step, the Autoencoder uses Stochastic Gradient Descent (SGD) to update the model parameters, such that it minimizes the reconstruction error, *i.e.*, the Mean Squared Error (MSE) (1) in our case, between $\mathbf{x}$ and $\hat{\mathbf{x}}$. The MSE is given as:

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

where N is the size of the input features and $\mathbf{x}_i[j]$ represents the j^{th} feature of data 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 [36]. 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 Chronos, 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, and iterate over the highest $\alpha\%$ of the reconstruction errors in steps of α/β, where α and β corresponding to the search depth and granularity, respectively. We select the corresponding reconstruction error as the current threshold and predict 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: α, β, $model$, $dataset$, $labels$
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 $optimal\_threshold$

Indeed, this requires a small labeled dataset. However, it does not undermine Autoencoder performance for zero-day attacks. As shown in Section V-C5, labels for known attacks across both attack types (*i.e.*, reflection- and exploitation-based attacks), are sufficient to detect unknown attacks (*i.e.*, attacks not used during threshold selection) that belong to the same attack types.

IV. Dataset Preparation

A. CICDDoS2019 dataset

We leverage the CICDDoS2019 dataset, which includes 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 (used in flow-based evaluation). 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.

TABLE I: The timing of attacks in the CICDDoS2019 dataset

Type	Attack	Time
Reflection-based	LDAP	March 11th, 10:21 - 10:30
	MSSQL	January 12th, 11:36 - 11:45
	NetBIOS	January 12th, 11:50 - 12:00
	PortMap	March 11th, 9:43 - 9:51
	SNMP	January 12th, 12:12 - 12:23
	SSDP	January 12th, 12:27 - 12:37
	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

TABLE II: Test dataset statistics

Attacks	Benign Packets (#)	Attack Packets (#)
LDAP	889	463,928
MSSQL	2,186	4,997,914
NetBIOS	14,291	1,582,576
PortMap	130,249	380,815
SNMP	1,910	4,998,090
SSDP	12,993	4,987,006
TFTP	13,447	4,986,553
UDP	15,390	4,984,610
UDPLag	3,166	3,123,705
SYN	4,863	3,775,195

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 243,708 packets in the benign dataset. In order to test the detection performance of the Autoencoder, we construct a separate test dataset for each of the 10 attacks that are shown in Table I. The test files, that we refer to as attack files, contain both benign and attack packets.

We use the IP address of the attack network to label the packets. Any packet that has the source or destination IP of the attack network 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management

V. Experiments

We carry out extensive experiments to evaluate the performance of our time-based anomaly detection system, *i.e.*, Chronos, compared to the 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 packets or flows for the time-based and flow-based anomaly detection, respectively. The Autoencoder must be re-trained and/or the threshold must be re-selected, whenever a performance degradation is noticed (*e.g.*, uncovering a high number of false positives). Note that identified anomalies must be further investigated by a security expert, or other methods (*e.g.*, using ML classifier [20]) that can identify the attack type and facilitate the subsequent stage of attack mitigation (*e.g.*, using reinforcement learning [37]).

In this section, we start by presenting the environment setup for experiments and the metrics employed for evaluation. This is followed by experiments on anomaly detection using time-based features, which includes: (i) justification for the choice of Autoencoder in Chronos, (ii) threshold selection using Algorithm 1, (iii) impact of different time-windows and their aggregations, (iv) comparison of Chronos to a state-of-the-art time-based system [15], (v) robustness to zero-days attacks, (vi) influence of limited training data, and (vii) effectiveness in the face of noise in training data. We conclude with a comparison of the time-based Autoencoder in Chronos to flow-based approaches [5], [20].

A. Environment setup

We perform our evaluation on a virtual machine (VM) deployed in an OpenStack [38] environment. The VM runs Ubuntu 18.04, managed by a KVM hypervisor. It includes 4 vCPUs, running on a server with 4x 64-bit Intel core processor CPUs. The VM also uses a GPU that features 28GB of RAM. Chronos is implemented using Python 3.7.6, while the Autoencoder leverages Keras library in Tensorflow 2.2.0 [39].

B. Evaluation metrics

We use the Autoencoder as a binary classifier to predict input instances (*i.e.*, packets or flows) belonging to the benign or the attack class. Accuracy, a widely used metric to evaluate classification performance, 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 other metrics to evaluate the Autoencoder prediction performance.

We denote the attacks as positive instances. Precision (2) is the proportion of attack predictions corresponding to the attack class. In contrast, recall (3) is the proportion of correct predictions for the attack class. A higher precision suggest lower false alarms, *i.e.*, benign instances being predicted as attacks, while a high recall implies that more attack instances are not missed.

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

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

However, similar to accuracy, precision and recall by themselves can not be used to evaluate a classifier's performance. For example, a classifier that predicts all input instances as attack will have a perfect recall. Similarly, a classifier that predicts all input instances as benign will have no false alarms. Therefore, the F1-score (4) is defined as a harmonic mean of precision and recall, which captures the trade-off between these metrics. Note that the harmonic mean is more useful than the arithmetic mean, since if either metric (*i.e.*, precision or recall) falls to zero, so would the F1-score.

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

Our heuristic (*cf.*, Section III-D) selects a threshold that maximizes the F1-score, our primary objective. However, the performance of the classifier (*i.e.*, the Autoencoder) is highly dependant on the selected threshold. Therefore, to showcase the performance independently of a specific threshold, we resort to the ROC curves, which represent the classifier performance in terms of the True Positive Rate (TPR) against the False Positive Rate (FPR) at various thresholds. We also highlight the Area Under the ROC Curve (AUC), which provides an aggregated measure of performance across the various thresholds. To compare the Autoencoder across all attacks, we use the mean AUC. An AUC of 1 depicts a perfect model, with TPR and FPR equal to 1 and 0, respectively.

C. Anomaly detection using time-based features

1) Autoencoder selection: Before we experiment with the time-based features to detect anomalies using Chronos, 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:

- $Arch.\ A = [k, k \times 30\%, k]$,
- $Arch.\ B = [k, k \times 70\%, k \times 30\%, k \times 70\%, k]$,
- $Arch.\ C = [k, k \times 80\%, k \times 50\%, k \times 30\%, k \times 50\%, k \times 80\%, k]$, 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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management

layer is 6. Therefore, for a given architecture, the number of neurons in the hidden layers changes with k, but the depth (*i.e.*, number of layers) for the Autoencoder remains the same. This impacts the number of trainable model parameters (*i.e.*, weights) relative to the input, and allows to evaluate the Autoencoder performance with the change in the number of input features. We leverage $k = 80$ by aggregating time-based features from different window sizes, denoted w (*cf.*, Section V-C3), 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 III. Note that during our experiments, we notice that the batch sizes affect the training time, without significant impact on detection performance.

TABLE III: 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

All architectures, except $Arch.\ D$ depict reasonable convergence within the first 50 epochs, as shown in Fig. 4. In contrast, $Arch.\ D$ takes the longest to converge, and continues to reduce the average MSE upto 200 epochs. Indeed, it is possible for $Arch.\ D$ to converge further to smaller average MSE. However, this comes with the highest number of model 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 remainder of our experiments.

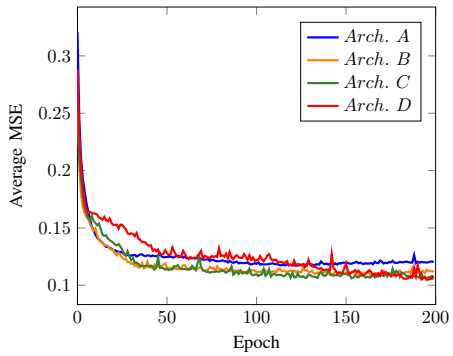


Fig. 4: Average MSE over training epochs with w=10sec, 1sec, 100ms, 10ms

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. To determine the threshold in Chronos, we randomly extract 1% of the packets from each attack file for the optimization dataset (*cf.*, Section III-D), while preserving the same proportion of attack and benign packets existing in that file. We observe that the outliers in the benign portion (*i.e.*, packets with high MSE) of the optimization dataset are minimal. Hence, the optimal threshold is typically amongst the first few reconstruction errors, as depicted in Fig. 5 (*i.e.*, the first in this case, highlighted in yellow). We employ hyper-parameters $\alpha = 20\%$ and $\beta = 20$ in our selection of the threshold. Both α (*i.e.*, search depth) and β (*i.e.*, search granularity) can be adjusted depending on the quality of optimization dataset, and allow for further exploration in threshold selection to potentially improve the Autoencoder performance in detecting anomalies. We discuss this further in Section V-C7.

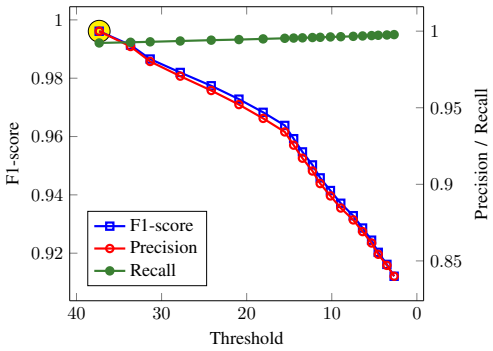


Fig. 5: Maximizing F1-score during threshold selection with $\alpha = 20\%$, $\beta = 20$, and w=10sec, 1sec, 100ms, 10ms, and the optimal F1-score highlighted

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 the training performance and showcase our threshold selection heuristic. The CICDDoS2019 dataset has traces of numerous DDoS attacks belonging to the well-known categories of 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 individual window sizes (*i.e.*, w=10sec, w=1sec, w=100ms, and w=10ms) in Chronos, with respect to anomaly detection across the different attacks.

Time-based features facilitate the Autoencoder in capturing the true characteristics of benign traffic. This is also supported by the low average reconstruction loss depicted in Fig. 4. As shown in Table IV, the Autoencoder is able to achieve significantly high F1-scores (*i.e.*, over 98.6%) for all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management

TABLE IV: Anomaly detection using time-based features and individual window sizes

	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
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
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

attacks with at least an individual window size. Furthermore, the precision is perfect for all attacks, except for PortMap, across all individual window sizes. However, the window size impacts the performance of the Autoencoder for some attacks (examples are highlighted in yellow in Table IV). For example, the 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 the best window size that allows to detect anomalies pertaining to the various attacks with high F1-scores, specifically for unknown attacks, is non-trivial. We show that aggregating time-based features across multiple window sizes in Chronos, achieves superior performance in anomaly detection across all attacks in the CICDDoS2019 dataset. However, note that the higher the number of aggregated window sizes, the more complex the Autoencoder with a higher number of model parameters to tune for convergence. Consider the chosen $Arch.\ C$ with an individual window size (*e.g.*, w=10sec). Each window size corresponds to 20 input features (*cf.*, Section III-B), resulting in $[20, 16, 10, 6, 10, 16, 20]$ architecture for the Autoencoder and 1,578 trainable parameters. Similarly, four window sizes aggregated together (*e.g.*, w=10sec, 1sec, 100ms, 10ms), results in $[80, 64, 40, 24, 40, 64, 80]$ Autoencoder architecture and 24,072 trainable parameters. While one would expect a better detection performance when the Autoencoder is trained on a larger number of features and trainable parameters, we show that this assumption does not hold. An Autoencoder that is trained on a smaller number of time-based features from a limited number of aggregated window sizes, provides comparable detection performance.

We evaluate the Autoencoder performance over multiple window size aggregations, *i.e.*, w=10sec, 1sec, 100ms, 10ms (*i.e.*, all window sizes in Table IV), w=10sec, 10ms, and w=60sec, 10sec, 1.5sec, 500ms, 100ms (*i.e.*, which is inspired by [15]). As shown in Fig. 6, the smaller aggregation across only two window sizes (*i.e.*, w=10sec, 10ms) outperforms the other window size aggregations. It achieves an F1-score of over 99% for most attacks and greater than 95% for all attacks, with an average of 99.2%. This is in contrast to the average F1-score of 99% and 96.8% for w=10sec, 1sec, 100ms, 10ms and w=60sec, 10sec, 1.5sec, 500ms, 100ms, respectively.

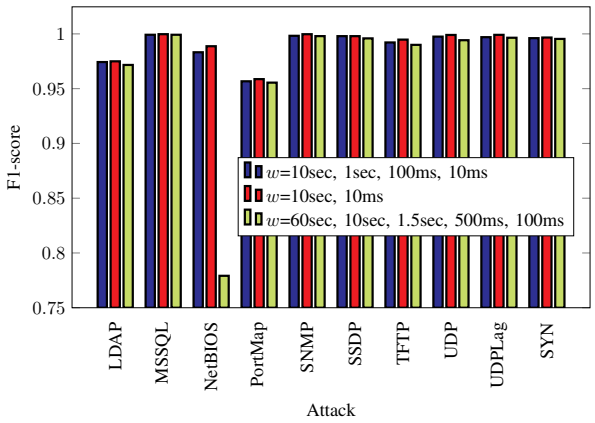


Fig. 6: Time-based anomaly detection using aggregated window sizes

Though remarkable performance is achieved for w=10sec, 10ms in Chronos, the lowest performing attack is PortMap, with similar performance across the other window aggregations. We partly attribute this to the peculiar nature of PortMap in the dataset, which we discuss further in Section V-C5. Furthermore, we notice a inferior performance of the Autoencoder for NetBIOS with the largest window aggregation (*i.e.*, w=60sec, 10sec, 1.5sec, 500ms, 100ms). This highlights that the window size aggregation and threshold selection play a quintessential role in the performance of the Autoencoder. Indeed, the threshold selection heuristic optimizes the average F1-score across all attacks (*cf.*, Fig. 6), but does not maximize the F1-score for an individual attack (*e.g.*, NetBIOS).

Hence, we compare the anomaly detection performance of the Autoencoder with different window size aggregations, independently of a specific threshold, as depicted in Fig. 7a, 7b, and 7c. The legend in the figures show the AUC of the corresponding ROC curve for each attack. We also showcase the mean ROC curve that represents the overall performance under all attacks. The Autoencoder has a similar performance for PortMap across the different window size aggregations for various thresholds, where w=10sec, 10ms marginally outperforms the other window size aggregations with an AUC of

0.9785. This is consistent with the results in Fig. 6.

Based on the ROC curves in Fig. 7c, we can conclude that the low F1-score of NetBIOS for w=60sec, 10sec, 1.5sec, 500ms, 100ms (*cf.*, Fig. 6) is due to the particular selection of threshold. More precisely, the selected threshold in Fig. 6 results in a perfect precision (*i.e.*, zero false alarms) and a low recall (*i.e.*, TPR = 0.64) for NetBIOS. However, as the threshold decreases, the recall approaches 1 without sacrificing precision, as shown in Fig. 7c. Similar to other window size aggregations, NetBIOS shows a high AUC of 0.9997 across various thresholds for w=60sec, 10sec, 1.5sec, 500ms, 100ms.

Furthermore, all window size aggregations perform well across all attacks. Table V provides a comparison for the smallest (*i.e.*, w=10sec, 10ms) and the largest (*i.e.*, w=60sec, 10sec, 1.5sec, 500ms, 100ms) window size aggregations, with respect to trainable parameters and the average training time of the Autoencoder. The result in Table V leverages $Arch.\ C$ (*cf.*, Section V-C1) and hyper-parameter settings in Table III, over the entire training dataset with 243,708 benign packets. For a fair comparison, the patience (*i.e.*, early stopping) hyper-parameter is ignored. Clearly, the mean AUC for w=10sec, 10ms (*cf.*, Fig. 7b) outperforms the other window aggregations with a smaller number of trainable parameters and a lower average training time with GPU acceleration. Furthermore, the feature extraction time for w=60sec, 10sec, 1.5sec, 500ms, 100ms is 60% higher than that of w=10sec, 10ms. Hence, we choose w=10sec, 10ms for the time-based Autoencoder in Chronos, and further experiment with it.

TABLE V: Autoencoder complexity and training overhead for aggregated window sizes

Window size (w)	No. of Trainable Parameters	Avg. Training Time (sec)
10sec, 10ms	6,116	1,190
60sec, 10sec, 1.5sec, 500ms, 100ms	37,490	1,583

4) Comparison to Kitsune: We also evaluate the efficacy of the anomaly detection pipeline presented in [15]. Recall that the authors propose their approach for an environment with resource constraint. To alleviate this constraint, they leverage an ensemble of light weight Autoencoders. They employ clustering to map the input features into smaller sets, one for each Autoencoder in the ensemble. This is followed by an output Autoencoder, which serves as a voting mechanism for the ensemble. The authors use a fixed window size aggregation of 60sec, 10sec, 1.5sec, 500ms, and 100ms, while allowing to tune the maximum number of inputs for the ensemble Autoencoders. This tuning can increase the speed of anomaly detection, at the cost of performance.

Fig. 8 illustrates the performance of their anomaly detection pipeline with respect to various DDoS attacks while setting the tunable parameter (*i.e.*, maximum number of inputs for the ensemble Autoencoders) to 10 (*i.e.*, default). Evidently, the anomaly detection pipeline in [15] achieves a comparable performance to Chronos, with a mean AUC of 0.9969. However, it is important to note that even a small difference in AUC can pertain to a high number of packet misclassifications. For example, with a specific threshold and corresponding high TPR of ≈0.95, there are ≈10,500 less false positives across all attacks in Fig. 7b (*i.e.*, Chronos) versus Fig. 8 (*i.e.*, [15]). Furthermore, the anomaly detection pipeline in [15] is rather complex. In contrast, Chronos leverages a single more complex Autoencoder that is able to correlate between a larger number of input features, and it marginally outperforms the anomaly detection pipeline in [15]. This shows the advantage of our approach in comparison to Kitsune, when deployed in an unconstrained, centralized environment, rather than on distributed IoT gateways with limited computing resources.

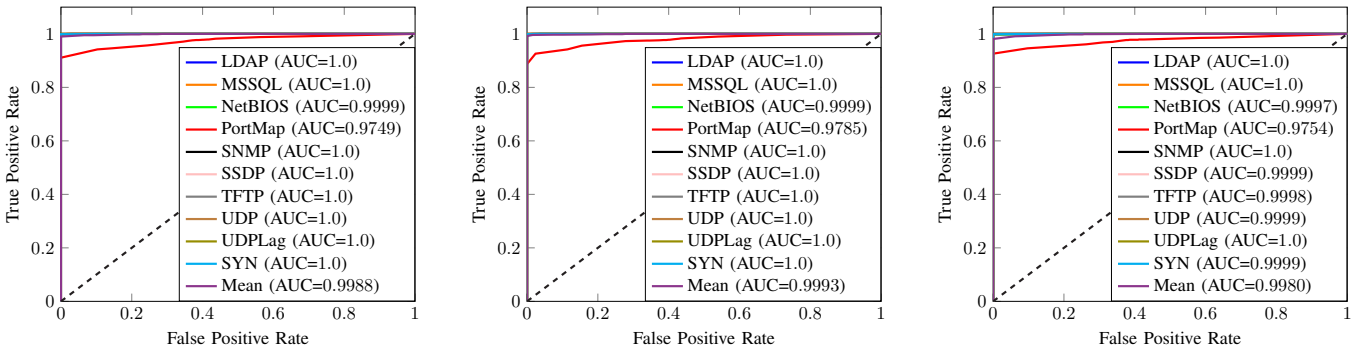
5) Robustness to zero-day attacks: Sophisticated adversaries often randomize malicious activities or leverage polymorphic attacks to avoid detection. Therefore, anomaly detection should be robust against unknown attacks or variations of known attacks. To evaluate the robustness of our time-based Autoencoder in Chronos, we modify the threshold selection, such that, we exclude some attacks (*i.e.*, NetBIOS, PortMap, and TCP SYN, belonging to both reflection- and exploiting-based attack types) from the optimization dataset. Hence, during threshold selection, the reconstruction errors depicting the characteristics of these *unknown* attacks (*i.e.*, NetBIOS, PortMap, and TCP SYN) do not influence the selection of the threshold. However, there are other known attacks, belonging to both attack types, leveraged during optimization, which can facilitate the detection of unknown DDoS attacks. As shown in Table VI, anomaly detection in the face of NetBIOS and TCP SYN attacks achieves an F1-score of over 99%. The performance against PortMap is just over 83%, which is primarily due to the high number of false alarms.

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
TFTP	0.99967	0.99802	0.99884
UDP	1.00000	0.99913	0.99957
UDPLag	1.00000	0.99965	0.99982
SYN	1.00000	0.99959	0.99979

The under performance of PortMap can be attributed to the selected threshold, as PortMap achieves a high AUC of 0.9785 in Fig. 7b. Regardless, we notice a slightly inferior performance for PortMap against other attacks across our experiments. This suggests that PortMap is rather peculiar and it is more complex to differentiate between PortMap and benign packets using time-based features. To validate this we portray a visual representation of the 40 time-based features of w=10sec, 10ms for different attacks. We leverage Uniform Manifold Approximation and Projection (UMAP) [40] to find a lower order (*i.e.*, 2-dimensional) representation of the features and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management



(a) w=10sec, 1sec, 100ms, 10ms (b) w=10sec, 10ms (Chronos) (c) w=60sec, 10sec, 1.5sec, 500ms, 100ms

Fig. 7: Anomaly detection ROC curves and AUC using time-based features and different aggregated window sizes

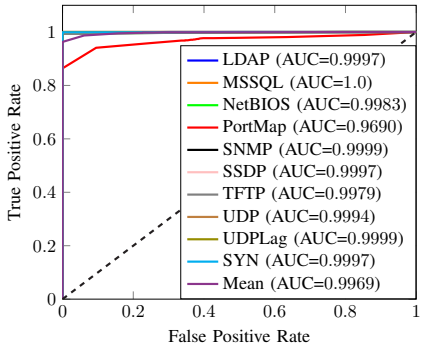


Fig. 8: ROC curves and AUC for the anomaly detection pipeline in [15]

plot them. Note that we use UMAP in the unsupervised mode when computing the lower order representation of each packet, *i.e.*, UMAP is unaware of a packet's label, benign or attack.

The results of UMAP on time-based features for benign and attack (*i.e.*, LDAP, PortMap, SNMP, UDPLag, NetBIOS, and TCP SYN) packets are visualized in Fig. 9. We randomly sample 100K benign packets and 50K attack packets from each attack, and visualize benign versus each attack separately. We observe that for most attacks (*e.g.*, LDAP, SNMP, and UDPLag), their lower order representation is well separated from benign packets. In contrast, a marginal overlap between benign and attack packets is depicted in the case of NetBIOS and TCP SYN attacks. However, the lower order representation of PortMap stands out with a significant overlap with benign packets. This visualization explains the inferior performance of time-based features in the detection of PortMap attack, in comparison to other attacks in the dataset.

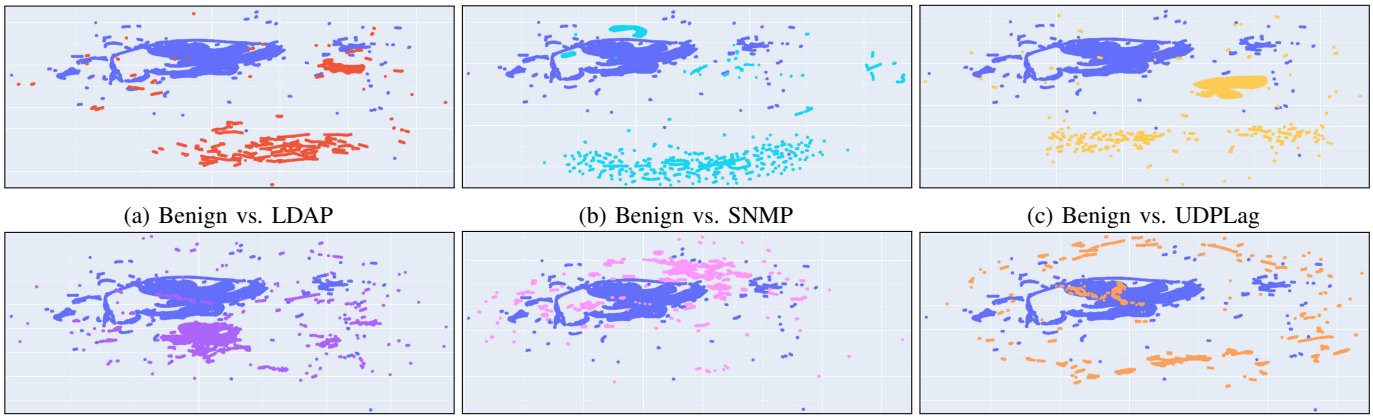
6) Training on partial data: In previous subsections, the time-based Autoencoder is trained on the entirety of benign dataset, which consists of 243,708 benign packets (*cf.*, Section IV). Indeed, training the Autoencoder on a large corpus of training data provides a holistic view of the benign traffic pattern, which can facilitate the model in achieving superior detection performance. Nonetheless, in this subsection, we evaluate the performance of Chronos in the absence of a large training dataset. Thus, we train the time-based Autoencoder on an increasing number of benign packets to showcase how quickly (*i.e.*, on a very small number of packets) the Autoencoder becomes very effective in detecting DDoS attacks. The results of our evaluation are depicted in Table VII.

TABLE VII: Autoencoder performance on partial, very small training data

w=10sec, 10ms				
Training Packets (%)	Training Packets (#)	Average Precision	Average Recall	Average F1-score
1	2,437	0.93091	0.91568	0.92206
2	4,874	0.91837	0.90900	0.91228
3	7,311	0.97147	0.93300	0.95050
4	9,748	0.99139	0.94797	0.96830
5	12,185	0.99789	0.96659	0.98160
6	14,622	0.99802	0.97431	0.98581
7	17,059	0.99798	0.97302	0.98510
8	19,496	0.99800	0.97178	0.98446
9	21,933	0.99778	0.96851	0.98260
10	24,370	0.99768	0.96511	0.98070
11	26,807	0.99770	0.95794	0.97676
12	29,244	0.99546	0.97477	0.98484

As shown in Table VII, we train the Autoencoder using only a portion (*i.e.*, in percentage) of the benign packets. Note that in an online setting, it is crucial to retrain the Autoencoder as new data becomes available. This can allow to account for changes in benign traffic pattern. Incremental learning [41] can be used to train the Autoencoder on new data, as it becomes available. However, in the absence of past data (*i.e.*, typically due to storage resource constraint), the model weights during training change based on the new data only, which can result in catastrophic forgetting and significant degradation in performance. Though, numerous approaches have been proposed to overcome this phenomenon (*e.g.*, [42], [43]), it is out of the scope of this paper to evaluate incremental learning and address its shortcomings. Hence, to alleviate the issue raised, when increasing the number of packets, we retrain the Autoencoder on the entire portion. For example, when training on 5% after training on 4% of the packets, the training dataset consists of the entire 5% of the benign packets, rather than the new additional 1% of packets only. As shown

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management



(a) Benign vs. LDAP (b) Benign vs. SNMP (c) Benign vs. UDPLag

(d) Benign vs. NetBIOS (e) Benign vs. TCP SYN (f) Benign vs. PortMap

Fig. 9: UMAP's 2-dimensional time-based features for benign and attack packets for w=10sec, 10ms

in Table VII, training the Autoencoder with only 1% of the packets (*i.e.*, 2,437) result in a respectable F1-score of 92.2%.

Recall that benign packets can act as outliers, such that their training MSE is high, even higher than some attack packets. As more benign packets are leveraged for training, the overall quality of benign packets may change, causing the Autoencoder to tune the model parameters accordingly. Therefore, as the Autoencoder is trained on more benign packets, we may see a slight drop in performance. For example, the average F1-score drops from 92.2% to 91.2% with 1% and 2% training packets, respectively. This can also be partly attributed to the selected threshold. However, the general trend is an increase in F1-score as more packets are leveraged to train the Autoencoder. Nevertheless, with only 6% of the benign packets (*i.e.*, 14,622), the Autoencoder achieves a high F1-score of almost 98.6%, as shown in Table VII. Clearly, with only a small number of benign packets, the Autoencoder achieves exceptional performance in anomaly detection across all attacks, making it suitable for deployment in an online setting with limited training data.

7) Training on noisy data: In an operational network, access to exclusively benign packets for training the Autoencoder is non-trivial. While in previous experiments, we train our time-based Autoencoder on exclusively benign packets, we show that, without loss of generality, the Autoencoder can be trained on a mix of unlabeled benign and attack (*i.e.*, in significantly smaller proportion) traffic. Therefore, in Fig. 10 we present the performance of Chronos, when the time-based Autoencoder is trained on benign data that is contaminated with noise (*i.e.*, attacks) from the test datasets in Table II. We ensure that the attack packets introduced in the training dataset do not overlap with the test packets. Fig. 10 depicts that as the traffic becomes noisier, the performance of the Autoencoder deteriorates. For noise levels of 0.4% and above, the average F1-score (*i.e.*, across all attacks) falls well below 60%.

Deep learning models, including Autoencoder, are prone to overfitting in the face of noisy training data. This can result in poor performance for anomaly detection on the attacks.

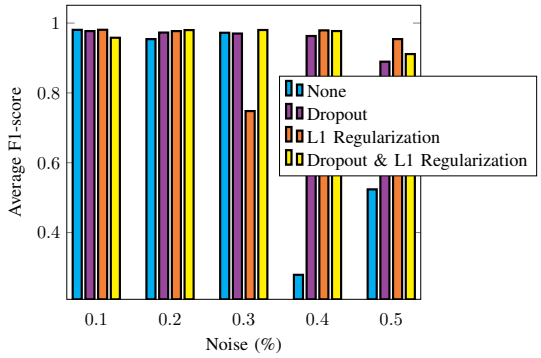


Fig. 10: Regularization on noisy training data for w=10sec, 10ms

However, regularization techniques can alleviate overfitting and improve model generalization. Dropout is a regularization technique, where a specified portion (*i.e.*, the dropout rate) of layer outputs are randomly ignored during each epoch, essentially training over multiple sub-architectures in parallel. L1 regularization penalizes the absolute value of the weights based on the specified regularization hyper-parameter λ. In L1 regularization, the weights can decay to zero, essentially compressing the model and alleviating overfitting. We evaluate dropout and L1 regularization with a dropout rate of 0.1 and λ as 0.01, respectively. We apply these regularization to the input layer of the Autoencoder. We also evaluate Autoencoder performance by combining dropout and L1 regularization, which has shown to improve model performance [44].

As shown in Fig. 10, both dropout and L1 regularization alleviate the impact of noisy training data on the Autoencoder performance in anomaly detection. The average F1-score with dropout and L1 regularization across the different noise levels and attacks is 95.4% and 92.7%, respectively. However, the Autoencoder performance using just dropout or L1 regularization, is rather erratic. For example, with 0.3% noise level in the training dataset, the Autoencoder with L1 regularization

significantly under performs in anomaly detection. This is alleviated with the combination of both regularization techniques. The Autoencoder with both dropout and L1 regularizations, generally outperforms the Autoencoder with just dropout, achieving an average F1-score of 96.1%. Furthermore, we perform an additional experiment for noisy training data at 0.5% (*i.e.*, with average F1-score of 91.1%), where we increase the hyper-parameters α and β in Algorithm 1 to 40% and 80, respectively. In this experiment, the time-based Autoencoder in Chronos achieves a high average F1-score of 97.5%.

D. Anomaly detection using flow-based features

Flow-based features have been widely adopted for DDoS detection. In this subsection, we gauge the performance of flow-based Autoencoder in detecting anomalies for various DDoS attacks. We leverage flow-based features from CICFlowMeter, which are available in CSV files as part of the CICDDoS2019 dataset. We evaluate two subset of flow-based features from CICFlowMeter. The first subset is inspired from Elsayed *et al.* [5]. The authors exclude some features to detect DDoS attacks, as they can introduce a bias in the model behavior. These features include source and destination IP addresses, source and destination port numbers, timestamp and flow ID. The second subset of flow-based features is inspired from Jia *et al.* [20], where RST flag count, PSH flag count and ECE flag count features are excluded, as they are less likely to impact classification performance [20]. We leverage $Arch\ .C$, *i.e.*, $[k, k \times 80\%, k \times 50\%, k \times 30\%, k \times 50\%, k \times 80\%, k]$, where $k = 79$ and $k = 76$ for [5] and [20], respectively. Note that the choice of the Autoencoder architecture does not have a significant impact on anomaly detection performance. For a fair comparison, we leverage the hyper-parameter settings in Table III, along with the threshold selection heuristic in Algorithm 1 with $\alpha = 20\%$ and $\beta = 20$.

The result in Fig. 11a shows a consistent performance across the feature sets from [5] and [20], with minor variations in F1-score for each attack. Both perform very well in terms of recall across all attacks, with an average of 99.9%, as shown in Table VIII. This is slightly better than the 98.4% recall in Chronos, *i.e.*, the Autoencoder with time-based features and w=10sec, 10ms (*cf.*, Fig. 6). However, the flow-based F1-score suffers with an average of 94.1% and 93.7% for [5] and [20], respectively. This is primarily attributed to low precision across all attacks using flow-based features (*cf.*, Table VIII), indicating a high number of false alarms. The average precision using flow-based features across all attacks is 89% and 88.2% for [5] and [20], respectively. This undermines the suitability of the Autoencoder using flow-based features to detect anomalies with respect to various DDoS attacks. In contract, the precision of the Autoencoder with time-based features and w=10sec, 10ms in Chronos, is perfect for all attacks and 99.99% for PortMap.

Consider the TCP SYN flooding attack that abuses the TCP three-way handshake procedure, and floods the server with repetitive SYN packets. Although both flow-based feature subsets employed in our evaluation include the flow's SYN flag count, among other flow statistics, the Autoencoder performs poorly in differentiating benign packets in the face of the SYN flooding attack. The precision for SYN attack is approximately 87% for both feature subsets. The ROC curves in Fig. 11b and Fig. 11c further assert the Autoencoder under performance for some attacks, irrespective of the selected threshold.

Regardless of the feature subset, the Autoencoder performance is impaired for most attacks, more so for the features in [20]. Evidently, the removal of the PSH, RST and ECE flag counts negatively impact Autoencoder performance, which contradicts the reason for removing these features in [20]. Nevertheless, the Autoencoder shows a lackluster performance using flow-based features with a mean AUC of 0.9738 and 0.9588 for [5] and [20], respectively. This is in stark contrast to the time-based features with w=10sec, 10ms in Chronos (*cf.*, Fig. 7b), where the Autoencoder achieves a very high mean AUC of 0.9993. Clearly, the flow-based features are not discriminative enough to distinguish between anomalous and benign behavior for various DDoS attacks.

TABLE VIII: Anomaly detection using flow-based features from Elsayed *et al.* [5] and Jia *et al.* [20]

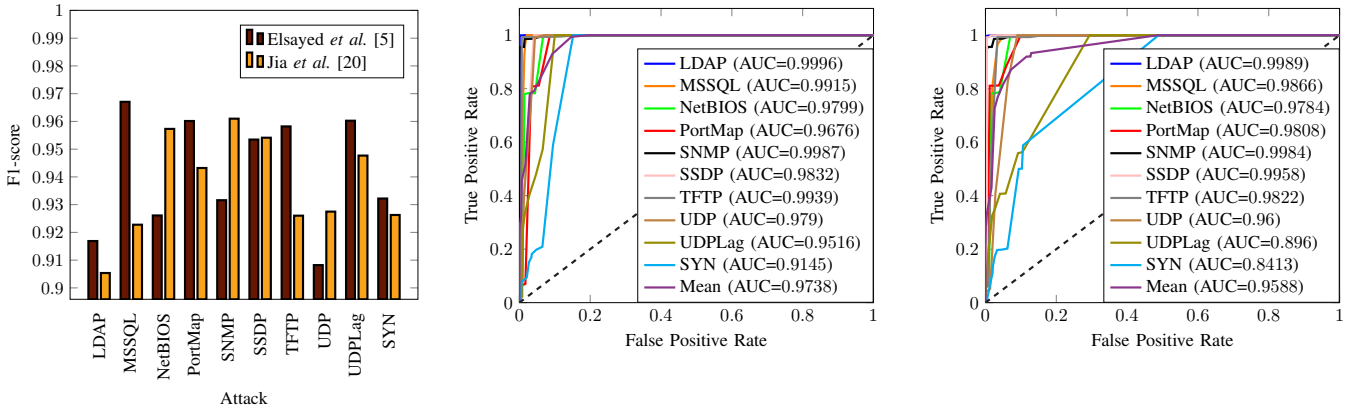
	Elsayed *et al.* [5]		Jia *et al.* [20]	
Attack	Precision	Recall	Precision	Recall
LDAP	0.84652	1.00000	0.82709	1.00000
MSSQL	0.93617	1.00000	0.85660	1.00000
NetBIOS	0.86236	1.00000	0.91806	1.00000
PortMap	0.92532	0.99765	0.89268	0.99977
SNMP	0.87194	1.00000	0.92550	0.99926
SSDP	0.91102	1.00000	0.91223	1.00000
TFTP	0.92421	0.99474	0.86288	0.99912
UDP	0.83186	1.00000	0.86477	1.00000
UDPLag	0.92376	0.99970	0.90078	0.99970
SYN	0.87326	0.99966	0.86292	0.99966

VI. Conclusion

We proposed Chronos, a novel time-based anomaly detection system that leverages an Autoencoder to detect anomalous DDoS traffic. We evaluated the impact of different window sizes in detecting anomalies pertaining to various DDoS attacks. We showed that by aggregating features across just two time-windows along with the proposed threshold selection heuristic, Chronos achieves a F1-score of over 99% for most attacks and greater than 95.86% for all attacks. Chronos also showed robustness to unknown attacks with an F1-score of over 99% for TCP SYN and NetBIOS, and over 83% for PortMap. We attributed and visualized the under performance of PortMap to its peculiar similarity to benign traffic. With the help of regularization techniques, Chronos alleviated the impact of noise in training data, and showed suitability for online deployment with superior anomaly detection performance on a very small number of packets.

Chronos also marginally outperformed [15] using a rather simple, less complex anomaly detection pipeline, while clearly out classing the flow-based features employed in [5], [20]. In the future, we will explore incremental learning to facilitate online detection of DDoS attacks. This will also entail adaptive

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2021.3088326, IEEE Transactions on Network and Service Management



(a) F1-score with flow-based features (b) ROC curves and AUC for features in [5] (c) ROC curves and AUC for features in [20]

Fig. 11: Autoencoder performance using flow-based features from Elsayed *et al.* [5] and Jia *et al.* [20]

threshold selection to account for changes in network traffic. Federated training of the time-based Autoencoder to facilitate knowledge sharing across the network edge is also a promising direction. Furthermore, we will investigate the efficacy of the time-based Autoencoder in differentiating between DDoS attacks and flash crowd traffic.

ACKNOWLEDGMENTS

We thank Dr. Makan Pourzandi, Dr. Stere Preda, Dr. Michael Liljenstam and Dr. Jakob Sternby from Ericsson Research, for their invaluable feedback. This work is supported in part by Ericsson Canada, and in part by the NSERC CRD Grant 536445-18.

REFERENCES

[1] Steve Morgan, "Cybercrime Facts and Statistics—Report: Cyberwarfare in the C-Suite," 2021, accessed: 2021-03-01. [Online]. Available: https://1c7fab3im83f5gqiow2qqs2k-wpengine.netdna-ssl.com/wp-content/uploads/2021/01/Cyberwarfare-2021-Report.pdf

[2] Rob Sobers, "134 Cybersecurity Statistics and Trends for 2021," 2021, accessed: 2021-02-28. [Online]. Available: https://www.varonis.com/blog/cybersecurity-statistics/

[3] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018.

[4] 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.

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

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

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

[8] 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.

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

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

[11] M. Kravchik and A. Shabtai, "Efficient cyber attack detection in industrial control systems using lightweight neural networks and pca," *IEEE Transactions on Dependable and Secure Computing*, 2021.

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

[13] H. Choi, M. Kim, G. Lee, and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5597–5621, 2019.

[14] Y. Intrator, G. Katz, and A. Shabtai, "Mdgan: Boosting anomaly detection using multi-discriminator generative adversarial networks," *arXiv preprint arXiv:1810.05221*, 2018.

[15] 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*, 2018.

[16] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[17] M. A. Salahuddin, M. F. Bari, H. A. Alameddine, V. Pourahmadi, and R. Boutaba, "Time-based anomaly detection using autoencoder," in *IFIP/IEEE International Conference on Network and Service Management*, 2020, pp. 1–9.

[18] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *IEEE International Carnahan Conference on Security Technology*, 2019, pp. 1–8.

[19] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *International conference on information systems security and privacy*, 2016, pp. 407–414.

[20] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, "Flowguard: an intelligent edge defense mechanism against iot ddos attacks," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9552–9562, 2020.

[21] M. Idhammad, K. Afdel, and M. Belouch, "Semi-supervised machine learning approach for ddos detection," *Applied Intelligence*, vol. 48, no. 10, pp. 3193–3208, 2018.

[22] Y. Gao, Y. Liu, Y. Jin, J. Chen, and H. Wu, "A novel semi-supervised learning approach for network intrusion detection on cloud-based robotic system," *IEEE Access*, vol. 6, pp. 50 927–50 938, 2018.

[23] J. J. Villalobos, I. Rodero, and M. Parashar, "An unsupervised approach for online detection and mitigation of high-rate ddos attacks based on an in-memory distributed graph using streaming data and analytics," in *IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, 2017, pp. 103–112.

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

[25] 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.
[26] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Computers & Security*, vol. 70, pp. 238–254, 2017.
[27] S. Maleki, S. Maleki, and N. R. Jennings, "Unsupervised anomaly detection with lstm autoencoders using statistical data-filtering," *Applied Soft Computing*, vol. 108, p. 107443, 2021.
[28] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *World Wide Web Conference*, 2018, pp. 187–196.
[29] S. Wambura, J. Huang, and H. Li, "Robust anomaly detection in feature-evolving time series," *The Computer Journal*, 2021.
[30] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *AAAI Conference on Artificial Intelligence*, 2021, pp. 2–9.
[31] The Wireshark team, "Wireshark, Go Deep," 2021, accessed: 2021-02-28. [Online]. Available: https://www.wireshark.org/
[32] ——, "The Wireshark Network Analyzer," 2021, accessed: 2021-02-28. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html
[33] Z. Yang, I. S. Bozchalooi, and E. Darve, "Regularized cycle consistent generative adversarial network for anomaly detection," *arXiv preprint arXiv:2001.06591*, 2020.
[34] 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.
[35] M. Wullink, G. C. Moura, M. Müller, and C. Hesselman, "Entrada: A high-performance network traffic data streaming warehouse," in *IEEE/IFIP Net. Operations and Mgmt. Symposium*, 2016, pp. 913–918.
[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[37] I. Akbari, E. Tahoun, M. A. Salahuddin, N. Limam, and R. Boutaba, "Atmos: Autonomous threat mitigation in sdn using reinforcement learning," in *IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.
[38] OpenStack, "Build the future of Open Infrastructure," 2021, accessed: 2021-02-28. [Online]. Available: https://www.openstack.org/
[39] Google Brain Team, "Tensorflow," 2021, accessed: 2021-02-28. [Online]. Available: https://www.tensorflow.org/
[40] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.
[41] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *European conference on computer vision*, 2018, pp. 233–248.
[42] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *National academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.
[43] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," *arXiv preprint arXiv:1703.08475*, 2017.
[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

Mohammad A. Salahuddin received the M.Sc. and Ph.D. degrees in Computer Science from Western Michigan University in 2003 and 2014, respectively. He was a postdoctoral research associate with the Université du Québec à Montréal and University of Waterloo, and a Visiting Scientist with Concordia University. He is currently a Research Assistant Professor of Computer Science at the University of Waterloo. His research interests include the Internet of Things, content delivery networks, network softwarization, network security, and cognitive network management. He serves as a TPC member for international conferences and is a reviewer for various journals and magazines.

Vahid Pourahmadi received the B.Sc. and M.Sc. degrees from Tehran University, and the Ph.D. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada. He served as a Postdoctoral Fellow at the ECE Department, University of Toronto, Canada, held a Technical Staff position at Blackberry Inc., Ottawa, and was a Wireless Research Engineer at the 5G group of Motorola Mobility, Chicago, IL, USA. His research interests include machine learning, wireless communication, and data analysis.

Hyame Assem Alameddine received her Ph.D. degree in Information and Systems Engineering from Concordia University, Canada in 2019. She holds a Master's degree in Computer Engineering - Information, Systems and Multimedia which she earned in 2015 from Conservatoire National des Arts et des Métiers (CNAM) University, France. She is an experienced researcher at Ericsson, Canada. Before joining Ericsson, she served as a postdoctoral fellow at the University of Waterloo, Canada in 2019 - 2020. She also worked as a programmer and application developer in multiple national and international companies between 2009 and 2014. Her current research interests include Network Function Virtualization, network security, 5G, internet of Things, cloud and edge computing. She serves as a TPC member for international conferences and a reviewer for various journals and magazines.

Md. Faizul Bari received his Ph.D. degree in computer science from the University of Waterloo, Canada. He completed M.Sc. in computer science and engineering from BUET, Bangladesh. After completing his Ph.D., Faizul worked as a Postdoctoral Research Fellow for two years at the School of Computer Science, University of Waterloo. After that, he worked as an R&D Software Engineer in the Distribute Database Lab at Huawei Canada Research Center. In 2021, he joined Spectrum S&C as the Chief Technology Officer. His research interests include network softwarization, cloud computing, blockchain, machine learning, deep learning, and cybersecurity.

Raouf Boutaba received the M.Sc. and Ph.D. degrees in computer science from Sorbonne University in 1990 and 1994, respectively. He is currently a University Chair Professor and the Director of the David R. Cheriton School of Computer science at the University of Waterloo (Canada). He also holds an INRIA International Chair in France. He is the founding Editor-in-Chief of the IEEE Transactions on Network and Service Management (2007-2010) and the current Editor-in-Chief of the IEEE Journal on Selected Areas in Communications. He is a fellow of the IEEE, the Engineering Institute of Canada, the Canadian Academy of Engineering, and the Royal Society of Canada.