IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 September 2023, accepted 5 October 2023, date of publication 23 October 2023, date of current version 1 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3327061

RESEARCH ARTICLE

IoT Network-Based Intrusion Detection Framework: A Solution to Process Ping Floods Originating From Embedded Devices

OMAR MOHAMMED ALMORABEA[1], TARIQ JAMIL SAIFULLAH KHANZADA[1,2], MUHAMMAD AHTISHAM ASLAM[3,1], FATHEAH AHMAD HENDI[4], AND AHMAD MOHAMMED ALMORABEA[1]

[1]Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 22254, Saudi Arabia
[2]Computer Systems Engineering Department, Mehran UET, Jamshoro, Jamshoro 76062, Pakistan
[3]Fraunhofer FOKUS, 10589 Berlin, Germany
[4]Department of Mathematics, King Abdulaziz University, Jeddah 22254, Saudi Arabia

Corresponding author: Omar Mohammed Almorabea (omohamedelmerabea@stu.kau.edu.sa)

The dataset can be found here: https://doi.org/10.5281/zenodo.811163. The code can be found here: https://doi.org/10.5281/zenodo.811222.

ABSTRACT Internet of things (IoT) devices are gaining traction around the globe. These devices are sometimes hijacked and turned into zombies or botnets. One risk posed by hijacked devices is a ping flood attack, also known as an internet control message protocol echo request flood. Current literature lacks a ping flood attack dataset generated from an IoT device. This paper contributes by developing an IoT network intrusion detection framework for ping flood attacks. This framework deploys an IoT testbed using embedded devices to emulate two datasets, normal ping traffic and malicious ping flood attack traffic. Features are extracted from the captured traffic using the Zeek tool. Attacks are detected using three machine learning algorithms: logistic regression, K-nearest neighbor, and support vector machine. These models are compared using evaluations such as the confusion matrix, accuracy, precision, recall, F1-score, and misclassification (error rate). The models are validated using split validation and cross-fold validation. The time consumed in training and testing the models across various data levels is also analyzed, along with the time required for feature extraction. The discrepancies between capturing tools are discussed. The use of criteria based on the time difference between requests to detect malicious traffic is considered, as is the impact of machine learning models on memory usage. Our work is compared with similar research. The testbed concluded that the K-nearest neighbor algorithm achieved 99.67% detection accuracy, with an error rate of 0.33% and an F1-score of 99.67%, which is the best amongst the three algorithms.

INDEX TERMS Flow information, ICMP flood, IoT, IoT devices, machine learning, ping flood, zeek.

I. INTRODUCTION

The technological renaissance is creating new paths for civilization to advance. The internet of things (IoT) is a new concept whose genesis could only be realized after recent developments in hardware, communication, and application layers [1], [2]. These layers are required for IoT devices to work. In its rudimentary sense, IoT technology enables remote physical devices that connect via various networks, mainly the internet, to exchange data and perform specific tasks. These tasks can involve collecting and retrieving data or manipulating the device's environment, i.e., interoperability between remote devices and performing desired goals.

The evolution of IoT security risks can be seen in the Mirai botnet attack, in which the attackers hijacked IoT embedded consumer devices. In [3], blame was placed on the conduct of both manufacturers and consumers. Manufacturers fabricated cheap devices with poor architecture design and code implementation, such as the teletype network protocol (Telnet). On the other hand, consumers failed to

The associate editor coordinating the review of this manuscript and approving it for publication was Cong Pu.

reset the default factory password. The Mirai attack targeted 15 000 high-level systems [4], and around 2.5 million IoT devices were infected [5].

One of the major types of attack that can suspend services is the denial of service (DoS) attack. This attack prevents legitimate users from accessing online services. This is done by a single device trying to overwhelm and suspend the operation of the service. A variant of DoS is distributed denial of service (DDoS), which is an orchestrated attack by many devices to simultaneously achieve the same goal as DoS. However, the ramification of DDoS is far greater, since DDoS attacks have many attackers from different networks.

An intrusion detection system (IDS) is used to detect anomalies and known malicious signatures at the network level. When an IDS detects malicious traffic, it generates alerts for a security team to investigate the incident. The signature-based approach searches incoming traffic for specific patterns that are known to be malicious. On the other hand, anomaly detection recognizes any abnormal behavior in network traffic. The IDS can be placed either inside or outside the network [6], [7], [8], [9].

The focus of this work is on the internet control message protocol (ICMP) echo request flood attack. The ICMP protocol is employed in the ''ping'' network utility to check the status of a remote device and obtain the round trip time in milliseconds. The attacker uses this utility to bombard the target with requests, and the target then has to process the request and reply to it. This consumes bandwidth and processing power of the target.

There are different structure headers and fields for each functionality of the ICMP protocol [10]. However, this work concentrates on ICMP echo request traffic; its structure is illustrated in Table 1. The data size in an ICMP packet depends on the operating system (OS): Microsoft Windows and Linux have different implementations. The former uses a data size of 32 bytes, while the latter uses 56 bytes. Linux divides its data into two sections: 8 bytes for a timestamp and the remaining 48 bytes for the rest of the data. The size of a ping packet in Windows is 74 bytes, while for Linux it is 98 bytes. Table 2 presents these differences. In our testbed, the Microsoft ping packet is used.

TABLE 1. ICMP echo request/response header structure.

Type Request=8, Response=0	Code Request=0, Response=0
Checksum Calculated when sent	Identification Coded by utility programmer
Sequence Calculated when sent	Data Device dependent

Zeek is a cybersecurity tool that extracts flow information by analyzing and summarizing traffic [11], [12]. The scripts used by Zeek run on packet capture (pcap) files to detect threats and malicious behaviors. Being a passive tool, Zeek does not have a countermeasure mechanism after detection.

TABLE 2. ICMP packet size breakdown.

#	Layer	Microsoft Windows OS size	Linux OS size
1	(Ethernet-Link Layer) header	14 bytes	14 bytes
2	(IP-Network Layer) header	20 bytes	20 bytes
3	(ICMP-Network Layer) header	8 bytes	8 bytes
4	ICMP data	32 bytes	56 bytes
		= 74 bytes	= 98 bytes

The lightweight transmission control protocol (TCP)/ internet protocol (IP) stack (LwIP) is a C language library that was created for embedded systems by Adam Dunkels [13], [14], [15]. This library is a lighter version of the traditional TCP/IP stack library present in OSs. It assists in reducing the random-access memory (RAM) usage while still providing the full TCP/IP stack functionality.

IoT devices are prone to hijacking as demonstrated by the Mirai attacks. This makes IoT devices a threat to internet services and network devices. One of the attacks employed in such a scenario is the ping flood attack, also known as ICMP echo request flood attack. This attack saturates the network bandwidth, computational processes, and memory of the targeted device. This leads to suspension of services to the legitimate users.

There are multiple variants of ICMP attack; however, this work focuses on an ICMP echo request flood attack with a spoofed IP address. This attack has a range of data sizes from 0 to 1472 bytes, i.e., not fragmented into multiple packets, and also where the packet configurations are normal, and not malformed.

There are multiple computer programs that can perform ping floods, such as hping3, Tribe Flood Network 2k edition (TFN2K), and Scapy [16], [17], [18], [19]. However, this work presents an attack program for embedded systems. Traditional computers usually have a higher processing speed and multi-threading capabilities.

The current literature lacks an accurate detection mechanism for ping flood attacks generated from IoT devices, i.e., embedded devices. In order to solve this problem, we have set three objectives:

- Creating a dataset that represents ICMP attack traffic generated from embedded devices (existing datasets are created using simulations and tools running on traditional computers).
- Finding the optimal features that represent ping flood attacks that are to be used in the machine learning (ML) prediction.
- Finding the optimal ML algorithm among the three chosen—logistic regression (LR), K-nearest neighbor (K-NN), and support vector machine (SVM)—that is best suited to detecting ping flood attacks.

In this work, an IoT network intrusion detection framework (INIDF) for detecting ping flood attacks generated from embedded devices is presented. This framework comprises four modules: a) knowledge building, in which information is gathered about the inner workings of other modules and

their performance, b) generating a dataset from IoT devices that emulates two scenarios of normal and malicious traffic, c) extracting features from the dataset, and d) implementing ML models with dataset injection for training and testing.

The dataset is generated from an espressif (ESP) 01S microcontroller for two scenarios: normal and malicious traffic. The features are extracted from network traffic using the Zeek tool. In the ML phase, training and testing is applied to LR, K-NN, and SVM algorithms. The performance of the models is measured using the confusion matrix and its derivative metrics.

Furthermore, the execution time is measured for the feature extraction module and the ML module (training and testing for all three algorithms). Memory profiling, also known as memory analysis, is also performed for the training and testing steps, with a Python script being used to evaluate and analyze the memory consumption of the applied ML algorithms. Finally, a hypothesis stated in previous research is argued to be disproven based on a real-world OS implementation.

There has been much research revolving around detecting DDoS attacks in IoT networks. However, after reviewing dozens of works, we found that none have ever created a ping flood attack from an embedded device, specifically a microcontroller (a special-purpose machine). This work fills this gap by 1) generating normal ping traffic and a malicious ping flood attack from a microcontroller, and 2) using only normal ICMP packets for the normal dataset—other works have used multi-protocol traffic in the normal dataset, making comparisons less valid. The traffic detection process in this work is then built on the knowledge and experience gained from the traffic generation process. That is, we create and manipulate ICMP packets, which in turn influence the feature extraction and the ML detection model.

The significance of this work is that the generated IoT data differ in volume (bandwidth) than the PC generated traffic. This is important because if the wrong bandwidth is used, then the ML model accuracy is diminished. Because if inaccurate data are used to train the model, then the model prediction will be inadequate. In other words, if the traffic bandwidth data is not correctly represented, then the subsequent steps like feature extraction and machine learning will be built on imprecise data.

Network traffic bandwidth is the maximum bit rate in a network system. Table 3 shows the bandwidth of a testbed experiment performed in the study to demonstrate the differences in bandwidth for three different devices (ESP8266, Raspberry Pi, and a PC). Each device generates a ping flood attack on a victim, the victim captures the data, and the bandwidth is extracted. As shown in Table 3, the ESP has the lowest bandwidth among the three, and the PC has the highest.

Bandwidth is an unreliable characteristic of networking, since networks rely on different physical factors to make a network connection, such as the type of connection (media and network interface card), the distance between devices, and the level of network congestion. As for wireless networking, it depends on the hardware and the IEEE 802.11 standard used. The experimental bandwidth measurement was extracted from a Wireshark I/O Graph for a single traffic instance, using an interval of 1 second. These statistics can fluctuate from device to device.

The ICMP or ping flood attack is the same across different devices, such as a server, PC, Raspberry Pi, or microcontroller. Because they use the same network protocol, any detection of ping flood attacks can be applied on any device. However, the volume of output data differs between device classes, and if the detection method uses machine learning, the model that is trained on the data is crucial to the accuracy of the system: if the data are provided from a different class, this would affect the outcome. This research provides a new dataset from a microcontroller, its bandwidth, and a new methodology. This work adds to the body of knowledge in that regard.

The contributions of this paper are as follows:

1) An emulated dataset, used to represent a ping flood, is extracted through a deployed IoT testbed.
2) Feature extraction is used to detect malicious ping floods.
3) An ML detection model is used to classify malicious and normal traffic in the deployed testbed.
4) The accuracy, efficiency, time consumption, and memory consumption of the system are analyzed for normal and abnormal traffic.
5) A previously stated hypothesis is validated.

The rest of the paper is organized as follows: Section II presents a review of the literature, Section III describes the experimental setup, Section IV outlines the INIDF, Section V presents and discusses the results, and Section VI concludes.

II. RELATED WORK

In [19], the researchers created a detection and prevention system for DDoS attacks based on a multiple-layer approach for an IoT-based environment. The detection of attacks can be performed online (cloud layer) or offline. The ML phase used a decision tree (DT) algorithm, and the prevention phase used a software-defined network (SDN) to drop malicious packets. The researchers deployed IoT devices in eight street lamp poles on a campus. The devices used various communication protocols, such as wireless fidelity (WiFi), ZigBee, Bluetooth, Ethernet, and long-range radio (LoRa).

The detection was performed for three DDoS attacks: ICMP flood, user datagram protocol (UDP) flood, and synchronize sequence number (SYN) flood. The available research datasets consist of comma-separated values (CSV) files and are found online [20]. The time consumption for online ML was greater than for offline prediction. The DT achieved over 97% for detection accuracy and F-score. The drawbacks of this research are that only one ML algorithm is used, and that it mentions that the sensors perform the attack, but does not explain this in great detail.

TABLE 3. Bandwidth measurement in testbed experiment.

	ESP 8266-WiFi	Raspberry Pi-WiFi	Raspberry Pi-Wired	PC-Kali-Wired
Bandwidth, bit/s (maximum bit rate)	918 784	5.164×10^6	5.1216×10^6	1.41416×10^7
Maximum number of packets per second	1552	10 785	10 670	36 406
ICMP echo request (ping) flood program used	Our code	hping3	hping3	hping3

Supervised deep learning (DL) approaches, such as convolutional neural networks (CNNs), were used to detect DDoS attacks in an edge computing environment in [16]. The experiment used a PC as an edge computing device to detect and distinguish traffic by using two-dimensional CNN models. The first dimension is for the packet traffic model and the second is for the packet feature model. The generated DDoS attacks were a SYN flood, a UDP flood, and an ICMP flood attack. The tool used to launch the attacks was TFN2K. The two-dimensional CNN models achieved over 99% detection accuracy. The measured time taken to divert the traffic to the edge device, perform the prediction, and relay the results was around 8 seconds. The disadvantage of this approach is the high processing time.

Numerous technologies were used in the creation of the IoT and industrial IoT (IIoT) datasets provided by [20]. After generating the datasets, they were used in a predictive analysis employing DL. This research created datasets for multiple attacks such as DDoS attacks, information gathering, man in the middle (MitM), injection, and malware attacks. The technology layers used in the creation of the datasets were cloud computing, network function virtualization (NFV), blockchain networks, fog computing, SDN layer, edge computing and IoT, and IIoT perception layer. There is a thorough review of current datasets, which are compared with their new IoT and IIoT datasets. A list of hardware, sensors, OS, and software used are provided. Moreover, a table of DDoS attacks and their generating tools is presented. The tools used for traffic capture and feature extraction were TShark and Zeek.

The number of selected features was 61 from a total of 1176 features. The ML and DL were implemented in the form of centralized and federated learning, respectively. The centralized approach used DT, a random forest (RF), SVM, K-NN, and a deep neural network (DNN). The federated approach used DL. The predictive models were used for binary, 6-class, and 15-class classification. DNN was the best-performing centralized approach for the 6- and 15-classes: it achieved 94.67% and 96.01% detection accuracies, respectively. Furthermore, RF, SVM, K-NN, and DNN achieved the same results for the binary classification with an accuracy of 99.99%. This research provides a comprehensive view and implementation of current technologies in creating new datasets for IoT and IIoT. However, their attacks were performed using PC tools.

The authors of [21] focused on detection of three DDoS attacks in an IoT environment—hypertext transfer protocol (HTTP) Get flood, TCP SYN, and UDP flood—using ML. The proposed framework generates and captures the traffic, and a feature engineering phase focuses on IoT-specific network behavior to determine the features, including both stateless and stateful features. The applied algorithms are K–NN, SVM, DT, RF, and a neural network (NN). All five algorithms achieved an accuracy higher than 99%. The drawback of this work is that the generated malicious dataset was generated using PC tools instead of real IoT devices. The generated malicious dataset was manipulated by spoofing the IP address with other IoT devices' IP addresses in the testbed.

Other researchers have focused solely on ICMP flood attacks, such as the authors of [22], who proposed detecting ICMP traffic using ML. The study generated a dataset containing normal and malicious traffic and analyzed the significant and relevant features of a ping flood in an IoT network. The traffic was detected using unsupervised learning algorithms like K-means clustering and a random tree (RT). Finally, a calculation of the energy consumption of their proposed detection model was provided. K-means and RT achieved a 99.94% detection accuracy for normal and malicious traffic. A shortcoming of this research is that the generated malicious traffic was produced using a traditional computer, not an embedded device. Their selected feature is discussed in the hypothesis validation section below.

In [23], the detection of four DDoS attacks in IoT networks using supervised learning is presented. These attacks include UDP flood, SYN flood, ICMP flood, and HTTP Get flood. The study used five consumer IoT devices to create the dataset. Their framework includes traffic generation together with scanning, a mitigation phase using rules along with policies procedures, features selection, and extraction of flow-dependent and flow-independent features. Finally, six ML algorithms were used: SVM, RF, DT, LR, K-NN, and naive Bayes (NB). The achieved detection accuracy for ML models ranges from 97%–99%, with RF as the best classifier. The shortcoming of this work is that the malicious dataset was generated using a tool from a traditional computer using Kali Linux utilities.

The authors of [24] focused on a different tactic for ICMP flood attacks, specifically the ICMP Smurf amplification attack. This differs from the traditional ICMP ping flood attack, which relies heavily on the echo requests. In contrast, the Smurf attack depends on echo replies to perform the attack. The detection method used for this research is the Kullback–Leibler divergence (KLD) metric. The feature used to detect Smurf attacks is the number of echo replies. The experiment was conducted with the DARPA 99 dataset. The research concluded that it can detect Smurf attacks for both

low and high rates of attack. However, the results did not address the accuracy of the suggested approach.

Authors in [25] applied an enhanced SVM to detect 9 different DDoS attacks, which included an ICMP flood attack. A dataset was generated for the testbed; however, this was not published. The performance of the ML was evaluated for linear, polynomial, and radial kernels. The radial basis kernel achieved the best prediction. A comparison between the created dataset and the KDDCUP 99 dataset was conducted. Although much research has been conducted on the KDDCUP dataset, they did not compare their results with other works. The dataset was created using a workstation and program scripts.

The researchers in [26] focused on detection of DDoS in software-defined IoT networks. The research considered detecting SYN flood, ping flood, UDP port scan, and UDP flood attacks. The ML algorithm used to detect the attacks was RF. The experiment used six sets of data features to predict the state of network traffic. The experiment took advantage of Minine-WiFi to simulate the software-defined IoT network, and the hping3 program was used to implement the attacks. The dataset obtained from the testbed has not been published online. The testbed simulated four scenarios with different settings, and results were presented for each. For example, in the first scenario, the ICMP flood detection accuracy was 0.9999. In the second and third scenarios, the corresponding accuracy score was 0.67. The research also compared its dataset with two other datasets. The results obtained suggest that if the number of trees or the tree depth for the RF algorithm is reduced, the accuracy and run time overhead will decrease as well.

The researchers in [27] applied an RF and a multi-layer perceptron (MLP) to the canadian institute for cybersecurity (CIC) IDS 2017 Dataset. This dataset does not include ICMP floods in its DDoS attacks. The features were obtained using the CICFlowmeter packet analyzer. The analyzer extracted 84 network features from the dataset. After the features were extracted, the waikato environment for knowledge analysis (WeKa) application was used to apply ML and a NN. The RF and MLP accuracies were 99.9% and 98.8%, respectively.

The authors of [28] developed a thresholding DDoS detection method. Four flow network features were selected to be the deciding factor in the proposed threshold algorithm. A dataset was created using hping3, Hyenae, and Metasploit. The new method was compared to DARPA 98 and DARPA 2000, and the detection accuracy was 99.5%.

The researchers in [29] developed a traffic flow classifier-based NN for detecting DDoS attacks in an SDN network. A mitigation process was also developed after detecting malicious traffic. The SDN environment was created using mininet and floodlight. The testbed generated three types of DDoS attacks: TCP flood, UDP flood, and ICMP ping of death attacks. The features used in the dataset are OpenFlow from the SDN network and raw packet attributes. The testbed utilized hping3 to generate the attacks. The detection accuracy achieved by the algorithm was 96.13%,

In [30], the researchers also created an SDN network using mininet and floodlight. The DDoS attacks generated by hping3 were TCP SYN flood, UDP flood, and ICMP flood attacks. The flow table attributes that are found in SDN networks were used as features in the dataset. The SVM algorithm was used in the detection process. The reported detection accuracy for ICMP flood attacks is between 90.48% and 95.24%.

In [31], the authors developed a DDoS detection framework in an SDN-IoT environment. The framework implemented three phases for detecting malicious traffic. The first phase used several ML algorithms, such as SVM, NB, RF, K-NN, and LR. The results from these algorithms were introduced to an ensemble voting algorithm to create an adaptive training model. This model was then used in a real live prediction. The experiment was constructed using Mininet-WiFi, while the SDN controllers used were floodlight and POX, and the Ping3 tool was used to flood the network. To measure the traffic, sFlow was used. The detection accuracy from the adaptive model is between 98% and 99%.

Detection and mitigation of low-rate DDoS (LR-DDoS) attacks in an SDN environment using ML technique was investigated by [32]. The research used ML algorithms such as J48, RT, REP Tree, RF, MLP, and SVM. A modularized approach was followed in the architectural design, in which an IDS system was installed on a different device and the intrusion prevention system (IPS) was integrated with the SDN controller. The dataset used for training was CIC DoS 2017; however, a testbed was deployed using mininet for testing purposes.

The LR-DDoS attacks included in this work were DDoS-Sim (a layer 7 DDoS simulator), GoldenEye, H.U.L.K. (the HTTP Unbreakable Load King), R.U.D.Y. (R U Dead Yet DoS tool), SlowBody2, Slow Headers, Slowloris, and Slowread. The innovation of the work lay in the framework and the implemented detection system. The traffic features were extracted using the flowtbag tool, and these features were then used in the IDS, i.e., the ML process. The best result reported was for the MLP algorithm with a detection accuracy of 95%. A shortcoming of this work is that there is no comparison between the obtained results and other research results; there was also no calculation of the time consumption of the ML process.

A. RESEARCH GAP

Tables 4–7 show an overview of the literature review. Although for some works, the ''research purpose'' column states that they generated attacks from IoT networks, in actuality, they were generated using greater processing power than is available in an IoT device.

Based on the above review, we can conclude that there is a research gap in generating a dataset of ICMP echo request

floods from embedded devices. To the best of our knowledge, no research has generated ICMP echo request flood and normal ICMP traffic from an embedded device. Researchers have either used existing datasets or generated a dataset using tools running on PCs. Available datasets have been inquired about; however, none are applicable to this work on ICMP echo request floods.

Traditional computers have higher processing speeds than embedded devices, which is reflected in their malicious attack rate through using multiple cores and threads. Researchers use PCs to generate ping flood attacks, then spoof the IP address to that of an IoT device.

This work presents an IoT network device that generates an ICMP echo request flood and normal echo request traffic. The ICMP traffic in the dataset is categorized as either normal or malicious by the use of ML algorithms. Furthermore, the performance is analyzed, according to various metrics, from the deployed framework.

III. EXPERIMENTAL SETUP

The characteristics of the devices used in the emulation were chosen based on their use in real-world IoT deployment. The choice of language was based on the low level of programming used for real-world IoT devices. The ESP device used in the testbed is called a system on a chip (SoC). It houses the microcontroller, RAM, input/output devices, and others on a single chip. Furthermore, it uses a WiFi module for network connection.

The ESP adapter device used in the testbed has two functionalities. The first provides power to the ESP-01S, and the second provides a universal serial bus (USB) to serial conversion for easy flashing and programming. The following subsection details the deployed testbed using embedded IoT devices for attack traffic analysis.

A. DEPLOYED TESTBED FOR TRAFFIC ATTACK ANALYSIS

This framework uses various devices to deploy the implemented testbed, depicted in Fig. 1, which are as follows:

- A PC, which plays multiple roles in the emulation process (victim/server/sniffer) and uses Ubuntu Linux OS on a virtual machine. The PC uses a category 6a copper wire for transmission.
- An ESP-01S device, used for generating genuine or malicious traffic. The media used for transmission is WiFi, with transmission frequency 2.4 GHz.
- An ESP-01S adapter used for easy ESP-01S programmability.
- A wireless access point (WAP), used to receive and transmit to/from the ESP-01S, with frequency band 2.4 GHz.
- A layer 2 switch.

The ESP was chosen for the testbed because it is used in a product known as Sonoff [38]. These products are used as switches for home appliances. For the current framework, C++ was used to program the ESP. Implementing the two

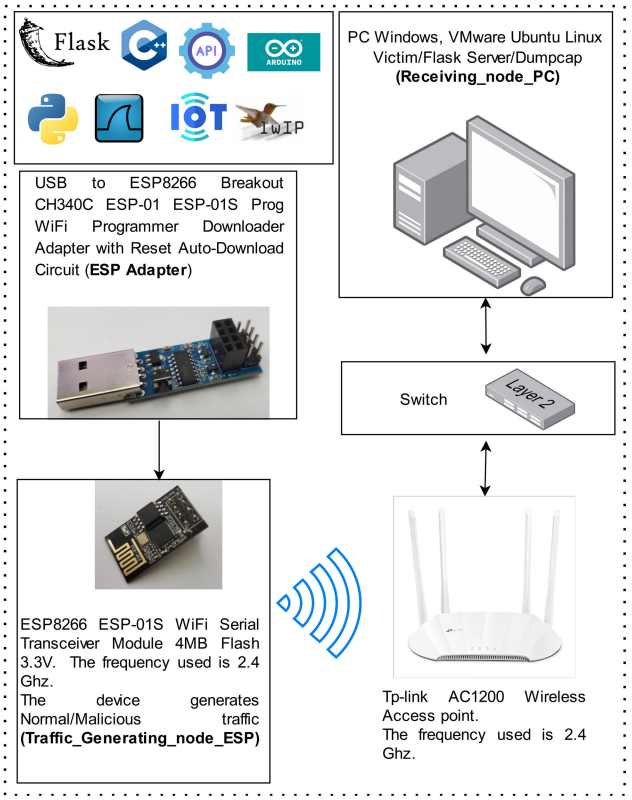

FIGURE 1. Network topology.

scenarios required the LwIP library for creating sockets, Ethernet headers, and IP headers [13], [14], [15]. The Flask microframework was installed on the PC to act as an API server for controlling the capturing process [39]. To capture and store traffic, Dumpcap was used [40]. The sequence of events is depicted in the diagram in Fig. 2, which shows generation of both normal and malicious traffic.

This process was automated; the normal and malicious datasets were generated at different timings with different pcap files. The steps are as follows:

1) The Traffic_Generating_node(ESP) sends a Get request with a flag to the Receiving_node(PC) to begin capture.
2) The Receiving_node(PC) sends a receive confirmation for the Get request.
3) The Receiving_node(PC) directs this message to the Flask server to begin the capture process.[1,2]
4) The Flask server runs the Dumpcap program to capture the traffic and store it in a pcap file.
5) The Traffic_Generating_node(ESP) sends normal/malicious traffic.
6) The Traffic_Generating_node(ESP) sends a Get request with a flag to the Receiving_node(PC) to close/end the capture.

[1]Screenshots of the Flask API can be found in [41].
[2]The Flask server code can be found in [42].

TABLE 4. Literature review summary table part 1.

Ref.	Year	Research Purpose	Dataset	Feature	Attack	Algorithm	Drawback			Contribution
							Initiation Device	Tool	ML Execution Time	
[33]	2020	A detection and prevention system for DDoS in two settings, online and offline, for an IoT-based environment; the prevention system used SDN to mitigate malicious traffic.	Found online [34]	Total of 7: packet length, timestamp, protocol, source IP, source MAC address, destination IP, destination MAC address.	ICMP flood, UDP flood, SYN flood	DT	Not mentioned	Not mentioned	×	Multi-layer traffic generation.
[16]	2022	A DDoS detection system using edge computing environment with deep learning. The detection is implemented on a data server for IoT, and is applied to a new generated dataset.	Not published	Total of 4: interval between two packets, sequence number, packet size, and transmission mode (class)	ICMP flood, UDP flood, SYN flood	CNN	No IoT network (Raspberry Pi was used)	TFN2K	×	Detection was computed in edge computing.
[20]	2022	Detection of various attacks in a new IoT and IIoT dataset created around multiple new technologies: cloud computing, NFV, blockchain, fog and edge computing, and SDN networks.	Found online [35]	Total of 61	DDoS (ICMP flood), information gathering, MitM, injection, and malware attacks	DT, RF, SVM, K-NN, DNN	No IoT network (PC was used)	hping3, slowhttptest, Nmap, Netcat, xprobe2, Nikto, Ettercap, xsser, sqlmap, metasploit, CeWL	×	Multiple attacks were detected.
[21]	2018	Detection of three DDoS attacks in an IoT environment using ML from a newly generated dataset.	Not published	Used stateless (packet header) and stateful (flow information) features. Stateless Features: packet size, inter-packet interval, protocol. Stateful features: bandwidth, IP destination address cardinality and novelty.	HTTP Get flood, SYN flood, UDP flood	K-NN, SVM, DT, RF, NN	No IoT network (PC was used)	Goldeneye, hping3	×	Considered stateless and stateful features.

TABLE 5. Literature review summary table part 2.

Ref.	Year	Research Purpose	Dataset	Feature	Attack	Algorithm	Drawback			Contribution
							Initiation Device	Tool	ML Execution Time	
[22]	2021	Detection of ICMP flood attack in IoT networks using pattern recognition.	[36]	Total of 12 but emphasized the detection by using two features (frame length and flags)	ICMP	RT, K-means	No IoT network (PC was used)	Not mentioned	×	Used unsupervised learning and investigated energy consumption.
[23]	2022	Detection of four DDoS attacks in an IoT environment using ML, through a framework that entails dataset generation, feature extraction, mitigation, and rules and policies.	Not published	Flow-dependent (stateful): IP address and port of source and destination, communication bandwidth, destination address count, new destination address count, transmission rate. Flow-independent (stateless): packet size, time interval between packets, protocol	UDP flood, SYN flood, ICMP flood, HTTP Get flood	SVM, RF, DT, LR, K-NN, NB	No IoT network (PC was used)	Goldeneye, hping3	×	Framework for DDoS detection in consumer IoT network.
[24]	2018	Detection of Smurf flooding attack by using Kullback–Leibler.	DARPA 99 [37]	Only 1 feature: number of ICMP echo replies	Smurf flood	Kullback–Leibler	No IoT network (PC was used)	Not mentioned	×	Applied Kullback–Leibler with Shewhart.
[25]	2011	Detection of nine different DDoS attacks by using ML and a newly generated dataset.	Not published	Total of 14	ICMP flood, UDP flood, TCP flood, Smurf flood, land flood, port flood, HTTP flood, session flood, IP flood	Enhanced SVM	No IoT network (PC was used)	Unnamed scripts	×	Used three kernels for enhanced SVM.

TABLE 6. Literature review summary table part 3.

Ref.	Year	Research Purpose	Dataset	Feature	Attack	Algorithm	Drawback			Contribution
							Initiation Device	Tool	ML Execution Time	
[26]	2020	Detection of DDoS attacks in a software-defined IoT network. The testbed was established on a new dataset.	Not published	Six different sets of Features	SYN flood, ping flood, UDP port scan, UDP flood	RF	No IoT network (PC was used)	hping3	×	Applied DDoS detection in an SDN environment.
[27]	2018	Applying ML predictive analysis on an existing dataset.	CIC IDS 2017	The CICFlowmeter tool extracted the features, 84 in total	The dataset did not include ICMP floods	RF, MLP	-	-	×	Applied ML and NNs to detect application layer DDoS.
[28]	2019	Detection using thresholding technique on a new dataset and comparing with DARPA 98 and DARPA 2000.	Not published	Total of 4: packet count, unique source IP, unique destination IP, unique protocol	Not formally mentioned	Thresholding algorithm	No IoT network (PC was used)	hping3, Hyenae	×	Applied dynamic thresholding for DDoS detection.
[29]	2020	Developed a traffic flow classifier-based neural network in an SDN network, also including a mitigation process.	Not published	OpenFlow attributes from the SDN network, and raw packet attribute (header information).	TCP flood, UDP flood, ICMP ping of death flood	Traffic flow classifier-based NN	No IoT network (PC was used)	hping3	×	Framework for detecting DDoS in SDN.

TABLE 7. Literature review summary table part 4.

Ref.	Year	Research Purpose	Dataset	Feature	Attack	Algorithm	Drawback			Contribution
							Initiation Device	Tool	ML Execution Time	
[30]	2018	Detection of DDoS attacks using ML in an SDN network.	Not published	OpenFlow attributes from the SDN network	TCP flood, UDP flood, ICMP flood	SVM	No IoT network (PC was used)	hping3	×	Applied ML for DDoS detection in an SDN environment.
[31]	2022	Developed a DDoS detection and mitigation framework for an SDN-IoT network. The framework entails ML prediction, ensemble voting, and a real live prediction system.	Not published	Rate of source IP, standard deviation of flow packets, standard deviation of flow bytes, rate of flow entries on switch, and the ratio of pair-flow entries on switch	ICMP flood	SVM, NB, RF, K-NN, LR	No IoT network (PC was used)	Ping3 (Python)	×	Applied multi-layer detection process.
[32]	2020	Detection and mitigation of low-rate DDoS attacks in an SDN environment using ML.	Training: CIC DoS 2017; Testbed: newly generated dataset	Flow information using flowtbag tool	DDoSSim, GoldenEye, H.U.L.K., R.U.D.Y., SlowBody, Slow Headers, Slowloris, and Slow Read	48, RT, REP Tree, RF, MLP, SVM	No IoT network (PC was used)	SlowHTTPTest	×	Applied multiple ML algorithms for detecting various DDoS attacks.
Our research	2023	Detection of ICMP flood originating from an embedded system by using ML	Created new IoT dataset	Duration, ICMP echo request	ICMP flood	SVM, LR, K-NN	Microcontroller	Created our own code in C++	✓	Added to the body of knowledge about IoT traffic volume. Created a new dataset for ICMP flood originating from an embedded device.

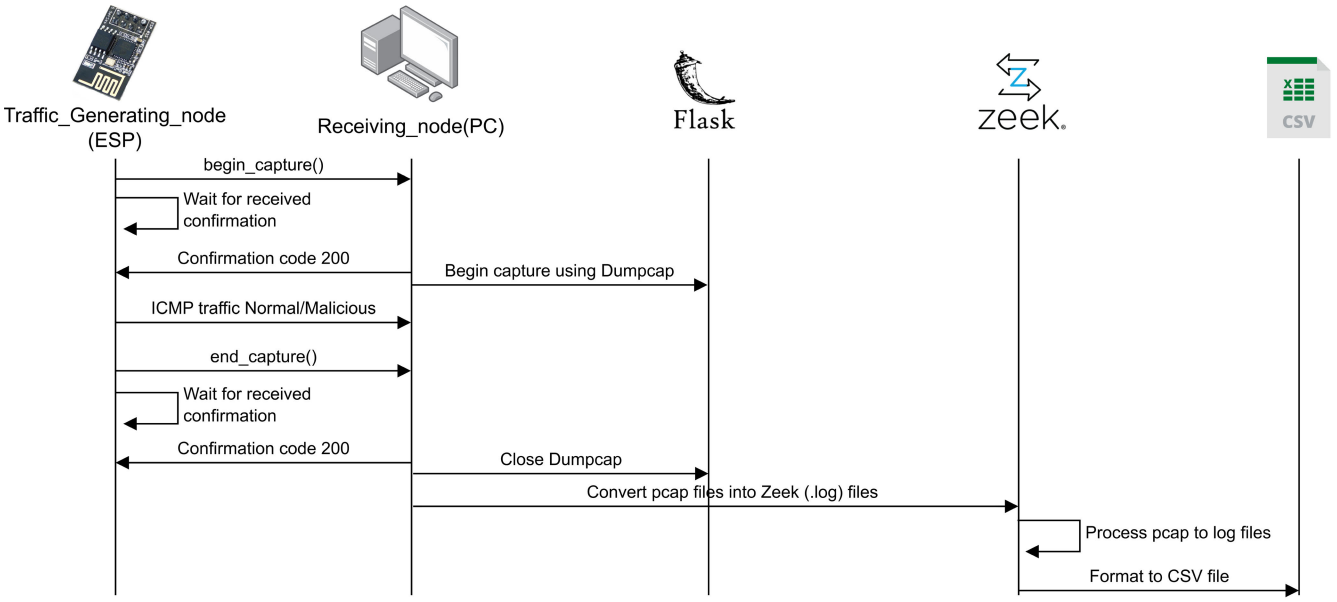

FIGURE 2. Sequence diagram for the process of packet creation and capturing into CSV files.

7) The Receiving_node(PC) sends a receive confirmation for the Get request.
8) The Receiving_node(PC) directs this message to the Flask server to close the capture process.
9) The Flask server closes the Dumpcap program to save the captured traffic in a pcap file.
10) The Receiving_node(PC) uses the Zeek tool to process the pcap files and outputs log files. The log files in Zeek are an overall abstraction (summary) of what traffic passed through the network.[3] A sample can be seen in Fig. 3.
11) After the log files are produced, the zeek-cut subtool and some other Linux programs are executed to filter and extract the summaries from the "conn.log" file.[4] The "conn.log" file shows the overall ICMP traffic for a sequential type of traffic.
12) The results produced in Step 11 are put into a CSV file. The normal and malicious traffic summaries are put into different CSV files.[5]

IV. INTERNET OF THINGS NETWORK INTRUSION DETECTION FRAMEWORK

This section outlines the development of INIDF for detecting ICMP echo request flood attacks generated from embedded devices. The overall architecture of the framework is illustrated in Fig. 4, showing the four modules:

- Knowledge building consists of four areas that were researched to build the framework.
- The generating data module involves configuring, managing, capturing, and storing the two scenarios of network traffic: normal and malicious traffic. There are many datasets for DoS and DDoS attacks generated using traditional computers; however, the focus of this work is to generate ICMP echo request flood attacks from embedded devices.
- The feature extraction module involves transforming the data and identifying useful ML features in the dataset.
- The ML module trains and tests three ML algorithms: SVM, LR, and K-NN.

In the following subsections, each module of INIDF is described in more detail.

An overview of the data generation, feature extraction, and ML modules is depicted in Fig. 5. As shown in this figure, the ESP device generates the normal and malicious network traffic. The traffic is captured and stored by Dumpcap, then the features (duration and received ICMP echo request) are extracted from the pcap and log files using the Zeek tool. Finally, the produced CSV dataset is injected into the ML algorithms for training and testing.

A. MODULE (1): KNOWLEDGE BUILDING

This module is the foundation of all other modules: it is the assembly of knowledge on various aspects of building and detection using prediction analysis. We have gone through the state-of-the-art literature resources to develop an understanding of previous models and their implementation.

This module does not take any input: it is the research one has to go through in order to generate the dataset, extrapolate its features, and predict its status as either normal or malicious. This step provides knowledge to the rest of the modules. We have gone through various learning and

[3]The conn.log files can be found in https://doi.org/10.5281/zenodo.8111635
[4]The script files can be found in https://doi.org/10.5281/zenodo.8112222
[5]The CSV files can be found in https://doi.org/10.5281/zenodo.8111635

	A	B	C	D	E	F	G	H	I	J
1	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	proto	service	duration	orig_bytes
2	1663647154.0397	CP6vTm3nEogRGpONjb	192.168.1.109	8	192.168.1.118	0	icmp	-	114.058119	4219936
3										

K	L	M	N	O	P	Q	R	S	T	U
resp_bytes	conn_state	local_orig	local_resp	missed_bytes	history	orig_pkts	orig_ip_bytes	resp_pkts	resp_ip_bytes	tunnel_parents
3737984	OTH	-	-	0	-	131873	7912380	116812	7008720	-

FIGURE 3. A sample of the information obtained from the Zeek "conn.log" file. The red columns are important features which are: protocol, duration, original packets received from sender, response packets replied to the sender.

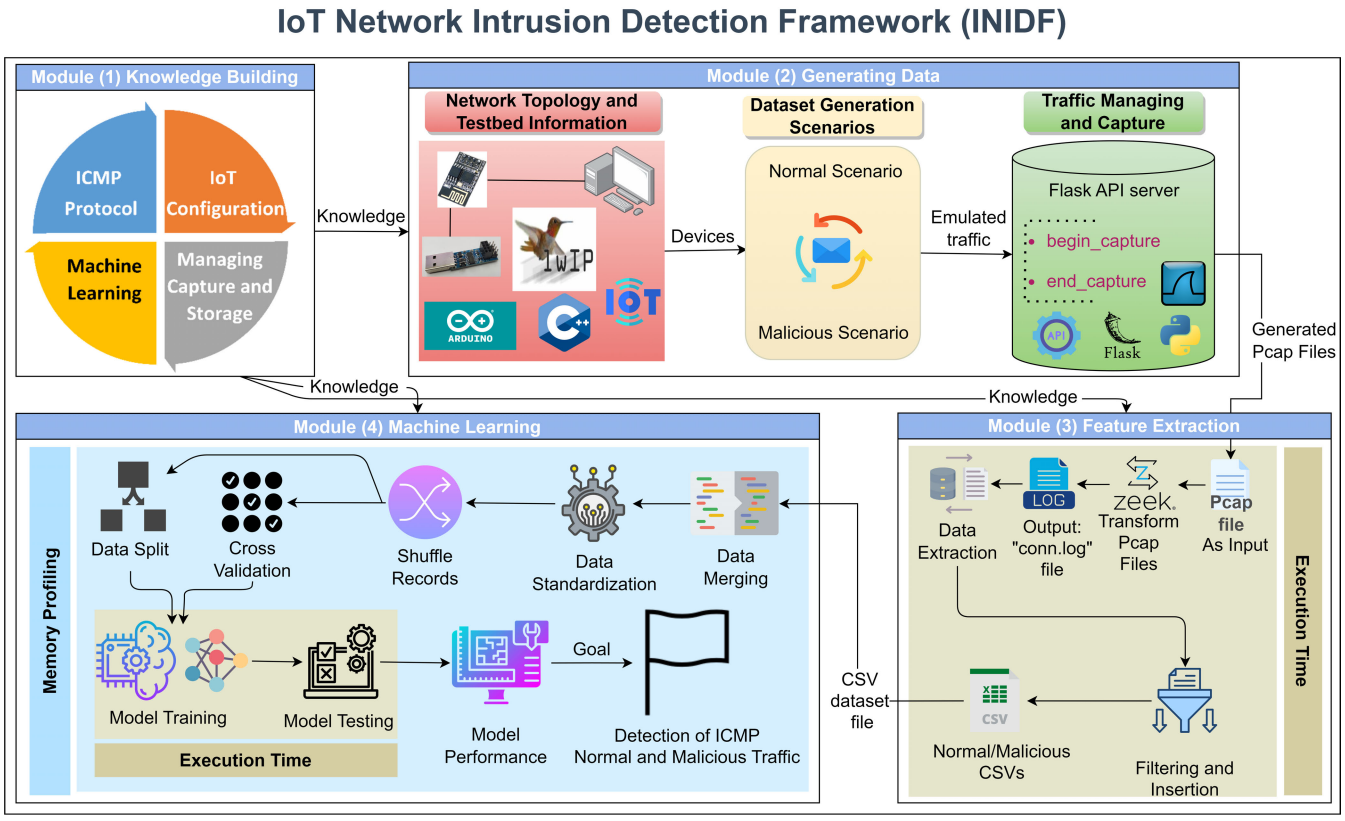

FIGURE 4. Architecture of IoT network intrusion detection framework (INIDF).

experimental steps through our journey to implement this framework. Table 8 lists all the learning steps and their categories.

The knowledge is divided into four categories: a) ICMP Protocol, b) IoT configuration, c) Managing Capture and Storage, and d) Machine learning implementations. All four categories forked into sub-steps that are carried out to develop the understanding and implementation. Every step acquired went toward building the rest of the modules.

B. MODULE (2): GENERATING DATA

The input to this module is a strategy derived through the first module, specifically, the ICMP Protocol, IoT Configuration, and Managing Capture and Storage categories. The information from these three categories helped in selecting the traffic-generating device, that is, the ESP 01S device. The ICMP (ping) code was programmed for normal and malicious scenarios on the embedded device (ESP 01S) using the C++ language. Coordination with the receiving end (the Python server in the PC) was performed using an API to manage capture and storage.

This phase was established because the current literature lacks ping flood data generated from embedded devices. The aim of this step is to emulate two scenarios, normal and malicious traffic, generated from real IoT devices. The following subsections discuss the dataset generation steps.

To create a normal or attack ping, a raw socket is needed in the code [43]. This raw socket is created from the LwIP stack

TABLE 8. Knowledge building steps.

Category	Steps
ICMP	ICMP request for comments (RFC) lookup. Literature review. Ping utility: various OS implementations. ICMP attack configuration.
IoT Configuration	Device Selection. C++ Programming. Embedded device application programming interface (API) in C++. LwIP stack library for embedded devices. Building ICMP configuration (ping utility) inside an embedded device.
Managing Capture and Storage	Python programming. Flask microframework for building API server. Dumpcap for capturing and storing pcap files.
Machine Learning Implementations	Feature extraction: • Extract flow information from pcap files. – Transform pcap to log file by using Zeek tool. • Extract features from "conn.log" file to CSV. – Use zeek-cut subtool. – Linux terminal text manipulation tools, e.g., awk, sed. Training and prediction: • ML algorithms. • Data standardization. • Using Python to implement ML. • K-fold cross-validation. • Stratified sampling. • Performance metrics. Memory profiling. Execution time.

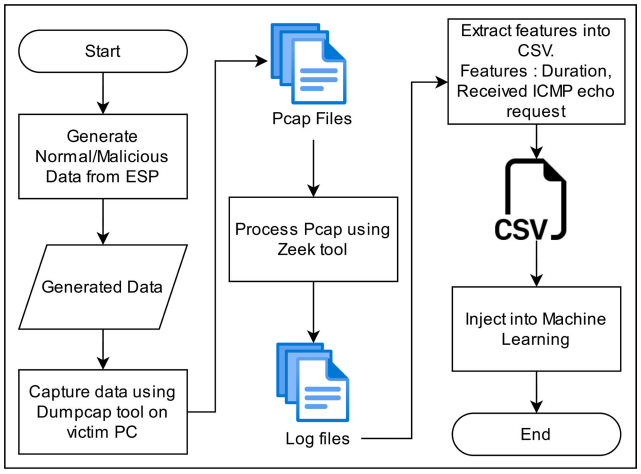

FIGURE 5. Breakdown of the data generation process.

that supports embedded devices. LwIP helps in providing the Ethernet and IP headers. However, we had to create the ICMP echo request header from scratch, i.e., all the fields in Table 1. In the normal scenario, we bind the socket to the IP of the network interface, i.e., the IP of the network. Conversely, in the malicious scenario, we bind the socket to a different IP each time a new packet is sent.

1) DATASET GENERATION SCENARIOS (EMULATION OF NORMAL ICMP TRAFFIC)

To generate normal ICMP traffic, the process followed by normal continuous ICMP traffic is illustrated in the sequence diagram shown in Fig. 6. This normal process was coded and configured for the ESP device to generate normal traffic from an embedded device. The normal experiment was repeated 500 times, producing 500 pcap files. The implementation of this normal process can be performed either by an embedded device or a PC because its process allows for a 1 second interval, thus, there is no dependence on heavy computational processes, unlike the malicious traffic. However, we wanted to be consistent with the implementation of the testbed, so it was implemented using an embedded device. The steps of the conventional ICMP request/response sequence, depicted in Fig. 6 are as follows:

a) PC_A sends an echo request to PC_B then awaits a response. This response can be a reply, an error, or nothing, with the latter resulting in "Request timeout."
b) PC_A waits one second regardless of response, then it sends another ICMP echo request to PC_B.
c) PC_B does not actually wait one second, however; it replies as soon as the request arrives, usually taking a fraction of a second.

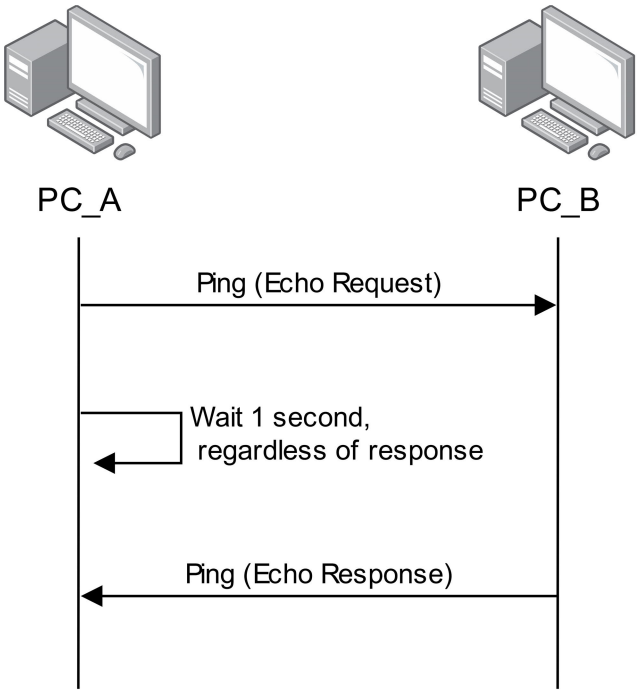

FIGURE 6. Normal ICMP request/response sequence.

2) DATASET GENERATION SCENARIOS (EMULATION OF MALICIOUS ICMP TRAFFIC)

The creation of attack traffic is automated, with the configuration of the attacks as follows. Each packet generated has a unique IP address, which is spoofed. The time duration is set as the condition for stopping the attack in the ESP device (attack device). The maximum time duration result achieved from the genuine dataset is 500 000 milliseconds. This value is set as the maximum time for the attack emulation, with the minimum duration set at one second. A random number between the maximum and minimum durations is generated by the ESP; this number is used as a condition for stopping the attack. The malicious experiment is repeated 500 times, producing 500 pcap files. This balances the genuine and malicious datasets.[6]

The normal and malicious traffic generation processes were developed using C++ and Python code.[7] The C++ code is uploaded to the ESP, while Python/Flask is used to control the capturing and storing process. This code is automated to create 500 pcap files for each scenario (normal and malicious). The emulation files and the contents of the normal and malicious ICMP traffic are shown in Table 9.

3) EVALUATION OF LEGITIMACY OF ATTACK DATASET

To evaluate the legitimacy of the produced attack dataset, a report [44] and paper [45] on detecting ping flood attacks using snort have been examined. In [44], a ping flood attack was identified if 500 packets were received in 3 seconds. In [45], a ping flood was defined as 400 packets being received in 4 seconds. We apply the definition of [44]. The produced attack testbed has achieved an average of 3071 packets in six instances of the attack in 3 seconds. This result surpasses the reported threshold of 500 packets in 3 seconds. Therefore, the established dataset has met the criterion and has been designated as a ping flood.

4) TRAFFIC MANAGEMENT AND CAPTURE

The framework applies the capturing phase through Dumpcap, a utility embedded in the Wireshark suite. The capture is triggered by a Get request made from the ESP to the Flask server. Afterward, the Flask server executes the Dumpcap program in a shell. Following that, the ESP reaches the end of its transmission in either the genuine or malicious scenario. It sends a Get request to the Flask server to end the capture. The Flask server closes off the capture/shell. There is a folder for each scenario, genuine and malicious.

Each folder contains 500 pcap files. Dumpcap is configured to only capture the attacker IP address because this work is only interested in detecting an ICMP packet from the ESP. Other traffic such as TCP and others will increase the pcap file size. Other research on DDoS attacks has tended to collect all traffic to detect general attacks associated with multiple protocols. The output of this module is the 500 pcap files for each of the normal and malicious scenarios.

Algorithm 1 describes the detection process in more detail. Algorithm 2 contains the pseudocode for generating malicious traffic, and Algorithm 3 for normal traffic. The capturing process of these two traffic generation routines is controlled by the Flask framework, as illustrated by Algorithm 4.

C. MODULE (3): FEATURE EXTRACTION

This module takes inputs from the first and second modules, the knowledge and the pcap files respectively. The knowledge gained from the first module is from the ML implementations category. The output of the second module is the set of pcap files, which are raw data from which features need to be extracted. The size of the captured pcap for the genuine dataset is 21.9 megabytes (MB), and the captured pcap for the malicious dataset is 17.2 gigabytes (GB). Both were captured over a duration of over 34 hours. From the experience of creating, experimenting with, configuring, and testing ICMP packets, it was concluded that the features of the ICMP packets cannot be trusted: the header information and payload can be manipulated. The reason for this phenomenon is that an ICMP packet is mutable. Any competent hacker could configure their own ICMP packet, similarly to how they were configured for this framework testbed. However, the flow information of traffic cannot be manipulated.

Flow information is the compendium of all traffic in a window of time, which summarizes the sequential traffic into one record. This record shows the protocol, service, duration, originating packets, and other information, as exhibited in

[6]The genuine and malicious datasets' pcap files can be found in https://doi.org/10.5281/zenodo.8111635

[7]The C++ code can be found in https://doi.org/10.5281/zenodo.8112222

TABLE 9. Files and packets content.

Normal Traffic			Malicious Traffic		
File Number	Number of Echo Requests	Number of Echo Responses	File Number	Number of Echo Requests	Number of Echo Responses
Number of echo requests = number of echo responses = file number Total files = 500			Randomly generated number of echo requests for each file, Number of echo responses for each file = variable (subject to computational power of target device) Total files = 500		

Algorithm 1 Traffic Detection

Input: Time: T_s = 1 sec, Packet Size: P_s = 74 bytes
Output: Detection of Normal and Malicious Traffic
$N_T = T_s * P_s$; /* N_T: Normal Transmission */
$M_T = (T_s * Random(1\,000 - 500\,000)) * P_s$; /* M_T: Malicious Transmission */
for i =1 to 500, increment by 1 $\{D_C \leftarrow (N_T, M_T)\}$; /* D_C: Data Capture (each scenario transmission has 500 time windows) */
$P_D \leftarrow D_C$; /* P_D: Processed Data (Tuple set) */
$MLA \leftarrow P_D$; /* MLA: Machine Learning Algorithm */
$P_M \leftarrow MLA$; /* P_M: Predictive Model */
P_M- applies (LR, K-NN, SVM), each algorithm calculates different P_M
$(0, 1) \leftarrow P_M$; /* Predicting Normal or Malicious traffic */
P_M is analyzed based on true positives, true negatives, false negatives, false positives, accuracy, precision, recall, F1-score, and misclassification.

Algorithm 2 Generating Malicious Traffic

SET serverName/IP to destination variable;
Function `sendSignal(signal)`**:**
  Build a GET request with signal value embedded in the request;
  SEND the GET request to destination;
  if *a response is not received such as response code 200* **then**
    DELAY for 2 seconds;
    CALL function sendSignal (signal) ;
    /* recursive call */
  end
return
Function `Main()`**:**
  SET timer to 0;
  SET counter to 1;
  while *counter less than or equal 500* **do**
    CALL function sendsignal (3);
    ; /* 3 indicates start capture of malicious traffic at the server side */
    DELAY for 2 seconds ;
    SET timer equals currentSystemTime plus random value from 1 second to 8.3 minutes;
    while *timer > currentSystemTime* **do**
      CALL function pingAttack (destination_IP);
    end
    DELAY for 1 sec;
    CALL function sendSignal(4) ; /* 4 indicates close capture of malicious traffic at the server side */
    INCREMENT counter;
  end
return

Fig. 3. Sequential traffic is the key here, because if time passes, the traffic is perceived as a new record. For example, if attack traffic is captured in one minute, it shows continuous bombardment of the victim, which results in a single record. The record shows the duration of communication between the two devices, and the number of received echo request packets. However, if there is a large time window of communication absence between packets, then it is perceived to be another record.

Because the value of the ICMP attack lies in its volume instead of its contents, unlike many other attacks, reviewing a single packet does nothing but legitimize the received packet. In essence, any individual ICMP attack packet is legitimate, but the quantity is problematic. Therefore, reviewing the traffic as a whole is necessary to distinguish its state; this is done using the Zeek tool, which summarizes sent and received traffic.

INIDF uses Zeek to read the pcap files and process them to create logs. Zeek has a sub-tool called zeek-cut, which performs extraction on the log files. This work uses a shell script that inputs pcap into Zeek for processing to log files, then extracts summaries from the log files using zeek-cut. These summaries are used as (piped) inputs to Linux programs such as awk and sed for filtering and insertion by using regular expressions [46], [47], [48], [49], [50]. The filtration is used for ICMP packets only, and the insertion part is used for labeling the data. Afterward, the results

Algorithm 3 Generating Genuine Traffic

```
COPY here lines 1–12 from the second algorithm;
Function Main():
    for ( i ← 1 to 500 by 1 ) {
        CALL function sendSignal(1) ;
        ;   /* 1 indicates start capture
         of genuine traffic at the
         server side */
        DELAY 3 seconds ;
        for ( j = 1; j ≤ i; j = j + 1 ) {
            PING(destinationIP);
            DELAY 1 second;
        }
        CALL function sendSignal(2);
        ;   /* 2 indicates close capture
         of genuine traffic at the
         server side */
    }
return
```

Algorithm 4 Flask Microframework (API Server Side)

```
Function killProcess(processObject):
    GET processObject.ID
    CALL function from the language library to kill
    the process by ID
return
Function Main():
    SET arg variable equal to the retrieved signal from
    the GET request
    SET process1 to an empty shell ;        /* for
    genuine traffic */
    SET process2 to an empty shell ;        /* for
    malicious traffic */
    if arg equals 1 then
        PRINT (“start genuine traffic”);
        SET cmd equal to a string “dumpcap -i 1 -f
        ‘ICMP’ -P -w output_destination”;
        PASS cmd string to process1 ;
        START process1 ;
    else if arg equals 2 then
        CALL function kill_process (process1);
        PRINT (“finished genuine traffic”);
    else if arg equals 3 then
        PRINT (“start malicious traffic”);
        SET cmd equal to a string “dumpcap -i 1 -f
          ‘ICMP’ -P -w output_destination”;
        PASS cmd string to process2;
        START process2;
    else if arg equals 4 then
        CALL function kill_process (process2);
        PRINT (“finished malicious traffic”);
    RETURN code 200 as a response
return
```

are inserted into a CSV file, to prepare them for the ML algorithms.

Two kinds of experiment were done using the Zeek tool; the first experiment was conducted for ping floods with a static IP address (not spoofed). The Zeek tool transforms the traffic flow to one record, showing the flow information for the whole time window, i.e., the duration and the sum of all echo requests. This is because one IP address is flooding the system. However, in the second experiment, ping floods with dynamic (spoofed) IP addresses were carried out.

In the second experiment, Zeek transforms each echo request into one record. Put differently, if there is a 1000 ping flood, the “conn.log” file (flow information file) shows 1000 records. This is due to each echo request having a different IP address. As a result, there is no cumulative duration provided nor a sum of echo requests, and thus calculating the duration and sum of echo requests needed to be handled. This rendered the Zeek tool useless. To overcome this hurdle, the duration needed to be calculated using the following formula:

$$T_\alpha - T_\beta = \Delta t$$

where T_α is the time of the last echo request, T_β is the time of the first echo request, and Δt is the duration.

The time mentioned in the formula is epoch time, which is in seconds. Epoch time determines the date and time for a computer clock, with the starting time varying from one OS to another. As mentioned, if there are 1000 echo requests, and each request is from a different IP address, then Zeek extracts 1000 flow records, and each record has its arrival epoch time attribute. This time attribute can also be found while inspecting a packet in Wireshark-Ethernet Frame-Epoch time. Another way of extracting the epoch time is by using the TShark tool, which exports it to a CSV file directly.

Despite that, we used the Zeek tool and extracted the time attribute from the “conn.log” file. This step required using bash scripting to handle the value storing and subtracting the time as shown in the duration formula. Additionally, the echo request tally was computed, after which, these two values were appended to the malicious CSV file. It should be noted that the “conn.log” file contains the records in disorderly fashion. Thereby, sorting needed to be performed first.

The features extracted from the conn.log file are the duration and number of originating packets known to be echo requests. The third column is the label, which is named “attack” and contains either one for an attack or zero for normal traffic. Ten-record samples of both CSVs are provided in Table 10. The dependent and independent features are illustrated in Table 11. The outputs of this module are the two CSV files, normal and malicious.

TABLE 10. Sample of CSV files for normal and malicious traffic.

No	Sample of CSV file for normal traffic			Sample of CSV file for malicious traffic		
	Duration (seconds)	Received echo requests	Attack (class)	Duration (seconds)	Received echo requests	Attack (class)
1	0.00002	1	0	1.80368	1327	1
2	1.00066	2	0	1.67734	1516	1
3	2.003343	3	0	2.14545	1653	1
4	3.004835	4	0	4.13569	2987	1
5	4.008014	5	0	5.01697	3221	1
6	5.007531	6	0	5.59779	4103	1
7	6.011577	7	0	8.91645	4326	1
8	7.007851	8	0	6.18157	4722	1
9	8.012637	9	0	9.73392	6367	1
10	9.015198	10	0	7.61751	6638	1

TABLE 11. Dependent and independent features.

Independent	Dependent
Duration	Attack
Number of Echo requests	

D. MODULE (4): MACHINE LEARNING DETECTION

This module receives inputs from the first module and the third module: knowledge and CSV files, respectively. The knowledge required here is how to train and test the modules. The CSV files with data on normal and malicious traffic are inserted into the ML process.

The data uses binomial/binary prediction, with normal traffic represented by "0" and malicious traffic by "1." There are three columns in the two CSV files: duration, number of originating packets (echo requests), and attack. The dataset does not have null or missing values. All values in the normal dataset are unique and do not follow the normal data distribution. On the other hand, the malicious dataset has a random data distribution.

The ML model predicts, on the basis of each record (time window), either normal or malicious traffic. The testbed uses Python and Jupyter to perform the ML process.[8] The Python version used in the experiment was 3.8.3, and the IPython version was 7.16.1, with jupyter_core 5.1.0. Several libraries were imported to be used in the ML process:

1) pandas.
2) time.
3) sklearn.linear_model.LogisticRegression.
4) sklearn.svm.SVC.
5) sklearn.neighbors.KNeighborsClassifier.
6) sklearn.preprocessing.MinMaxScaler.
7) sklearn.model_selection.train_test_split.
8) sklearn.metrics.
9) sklearn.model_selection.cross_validate.

The pandas library reads the two CSV files (normal and malicious) and places them in separate variables. A number_range array variable, containing the values 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500, is used to increment the size of records/observation for each scenario to see the trend of performance and execution time.

A for loop iterates through the array and assigns records from each dataset to the number_range current iteration. For example, in the first iteration, 50 records are placed for each scenario, normal and malicious, then these two variables are combined into one variable that has a total of 100 records (normal and malicious). Thus, the data have a balanced distribution.

Next, the data are segregated into dependent and independent variables, which are passed to the train_test_split method to be split into training and test data in a 70:30 ratio, respectively. The split validation shuffles the data using a fixed seed of value 177, and uses stratified sampling to ensure a balanced dataset, i.e., a 50%:50% ratio of normal and malicious traffic.

The training data are scaled using the MinMaxScaler, which transforms them to values between 0 and 1. Since the malicious dataset has very large values compared to the normal dataset, scaling is necessary for accurate detection. The scaling first obtains the minimum and maximum values from each feature and then applies the scaling formula to each field. The MinMaxScaler formula [51] is as follows:

$$Scaled\ Dataset = \frac{D - D.min}{D.max - D.min}$$

Next, each model is trained on the data. The test data are also scaled based on the minimum and maximum values of the training data. Subsequently, the test data are predicted and compared to their known classes. Afterward, the performance is evaluated based on comparing the confusion matrix and its derivatives. Cross-validation is applied to the dataset with 5 and 10 K-folds, and also with stratified sampling. After the normal and malicious datasets are merged, the independent data are scaled as a whole, and cross-validation is applied.

Pattern recognition is a type of ML technique that enables more advanced learning. A model trained with data can perform better than statistical techniques. The three supervised classification algorithms used in this framework

[8]The Jupyter ML file can be found in https://doi.org/10.5281/zenodo.8112222

are LR, K-NN, and SVM. Referring to [20], [21], and [23], these algorithms were chosen based on their high detection accuracy.

1) SUPPORT VECTOR MACHINE

SVM is a supervised ML algorithm that can be used for both classification and regression problems. The classifier is non-probabilistic and uses a hyperplane with maximum margin, which is calculated using a convex optimization problem [52], [53], [54]. The SVM linear mathematical model is [55]

$$min_{\omega,b,\zeta} \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\zeta i$$
$$y_i(\omega^T\phi(x_i) + b) \geq 1 - \zeta_i$$
$$\zeta_i \geq 0, \quad i = 1, \ldots, n$$

2) LOGISTIC REGRESSION

The LR algorithm computes a value ranging from 0 to 1 from any input value by using the sigmoid function. A cutoff or threshold is placed at 0.5 to distinguish between the binary classes. Finally, the produced probability classifies which side of the line the value lies on [56]. LR is sensitive to outliers. The LR mathematical model is [57]

$$p(X_i) = \text{expit}(X_i\omega + \omega_0) = \frac{1}{1 + \exp(-X_i\omega - \omega_0)}$$

3) K-NEAREST NEIGHBOR

The K-NN algorithm calculates the distance from a given point x to all points in the data. Afterward, the data points are sorted in order of increasing distance from x. Finally, the majority label of the K closest points is predicted [58]. The optimal K parameter in the testbed is 5. The K-NN mathematical model is [59]

$$\arg_L \max \sum_{i=0}^{N-1} \zeta p_i$$

E. HYPOTHESIS VALIDATION THROUGH A RECURRENT EXPERIMENT

While researching and experimenting it was found that a hypothesis claimed in [22] is not true. The work in [22] created a dataset for normal and malicious traffic for ping flooding detection. After analyzing both datasets, the authors concluded that an ICMP packet can be identified by two features: frame size and IP flag. Table 12 shows the features and a rebuttal verified by the testbeds implemented under this framework.

V. RESULTS, ANALYSIS, AND DISCUSSION

The three ML models are evaluated using the confusion matrix. This is used in classification problems to assess the locations of errors in the model. There are four cells in the confusion matrix to identify the whereabouts of the mistakes.

The true negative (TN) rate describes traffic that is correctly predicted as normal ICMP traffic. The false positive (FP) rate, sometimes called type 1 error, describes normal ICMP traffic that is incorrectly predicted as ping flood traffic. The false negative (FN) rate, also known as type 2 error, describes ping flood traffic that is incorrectly predicted as normal ICMP traffic. The true positive (TP) rate represents ping flood packets that are correctly predicted. Table 13 shows the obtained confusion matrix. There are five performance metrics that can be derived from the confusion matrix, as listed below:

- The F-score measures a model's accuracy on a dataset:

$$2 \times \frac{Precision \times Sensitivity}{Precision + Sensitivity}$$

- The sensitivity (recall) measures the percentage of predicted positives among all positive cases:

$$\frac{TP}{TP + FN} \times 100$$

- The precision measures the percentage of actual positives among predicted positives.

$$\frac{TP}{TP + FP} \times 100$$

- The misclassification (error rate) measures the percentage of observations that are incorrectly predicted:

$$\frac{FP + FN}{Total\ Predictions} \times 100$$

- The accuracy measures how often the model is correct:

$$\frac{TP + TN}{Total\ Predictions} \times 100$$

The time taken for training and testing was measured using Python's time.perf_counter method, and the results are shown in Fig. 7. The results of testing the models in split validation, using the confusion matrix and its derivative performance metrics, can be seen in Figs. 8–11. The cross-validation result is illustrated in Fig. 12.

These measurements were carried out on a PC using 64-bit Microsoft Windows 10, running on an Intel i7-7700K CPU at 4.20 GHz with 16 GB RAM. The best training time was achieved by K-NN; however, the best testing time was achieved by the LR algorithm. The SVM algorithm required intermediate amounts of time.

The bar chart in Fig. 7 illustrates the time consumed in training and testing the three algorithms across data increments. These data increments are for split validation, that is, all dataset increments are divided into 70:30 training:testing ratios. First, we discuss the training execution times. Overall, as can be seen in Fig. 7, the execution times for training LR and SVM are higher than for K-NN, with LR taking the longest time. One can observe that the time to train SVM increases steadily with the data size. The growth percentage of SVM from its initial 10% to 100% is 356.297%.

$$Growth\ Percentage = \frac{presentValue - pastValue}{pastValue} \times 100$$

TABLE 12. Hypothesis validation.

Type	Attributes (claimed by research [22])		Rebuttal
	Frame length (bytes)	Flags	
Normal ICMP	98	0x02	The 98-byte size is for a Linux system as mentioned in Section I. The frame length and the data size are positively correlated. The problem with this hypothesis is that Microsoft Windows produces an ICMP packet with a frame size of 74 bytes. In an ICMP packet, the payload can have a value from 0 to 1472 bytes without fragmentation. The data of an ICMP packet can be configured to any number in the above range. This causes the frame size feature to be unreliable. The 0x02 flag represents more fragments [60]; however, the article did not mention anything about fragmentation.
Ping Flood	42	0x00	The 42-byte size indicates that the data size is 0 bytes. The flag attribute of 0x00 is for IP fragmentation, which indicates there is no fragmentation set [60]. The flag 0x00 is in a normal Microsoft Windows ICMP echo request packet. The detection of ping floods with these features is not applicable to real-world data.

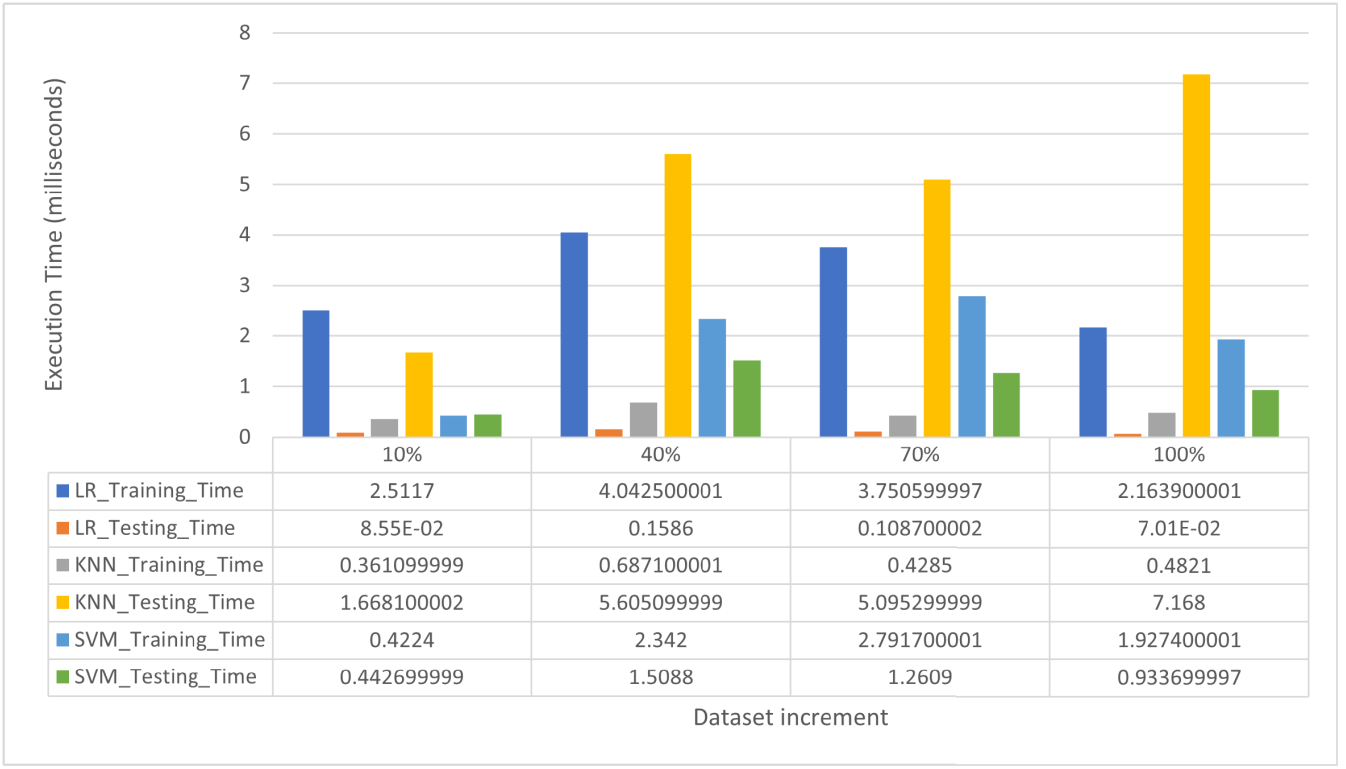

FIGURE 7. Execution time for training and testing of three algorithms over data increments for split validation.

TABLE 13. Confusion matrix.

700 trained / 300 tested	
TN: (LR-150), (K-NN-150), (SVM-150)	FP: (LR-0), (K-NN-0), (SVM-0)
FN: (LR-7), (K-NN-1), (SVM-9)	TP: (LR-143) (K-NN-149), (SVM-141)

As for the testing execution times, generally, K-NN and SVM consumed more testing time than LR, as can be seen in Fig. 7. In contrast to the LR training time, it requires little testing time. This means that the data size and proportion do not affect the LR algorithm. Unlike K-NN training, the K-NN testing time inconsistently rose, with a growth rate of 329.736%, which makes it the highest among the other two. SVM also gradually increases in testing time, but at an acute angle, unlike K-NN.

Figs. 8-11 show the performance metrics for the three models in 20% data increments. This measurement is for 70:30 split validation. In Fig. 8, we can observe that K-NN began with 93.33% detection accuracy at the 10% increment, after which the data size increased and the accuracy began steadily increasing. K-NN achieved the best detection accuracy among the three models at 99.67%. In contrast, SVM achieved the lowest with 97% detection accuracy. LR scored a little higher than SVM at 97.67%.

In Fig. 9 misclassification, i.e., the error rate, is depicted for three ML models over a 20% increment of the dataset. Misclassification is the complement of accuracy, so it is high when accuracy is low. K-NN has the lowest error rate at 0.33%.

Fig. 10 demonstrates how often the models fail to spot malicious traffic. Because all precision values are 100%, only the recall values affect the accuracy. If the accuracy and precision are examined side by side, they have the same distribution but at different ranges. K-NN achieved the highest recall, meaning that it was the most accurate.

Fig. 11 illustrates the F-score for all three ML models in 20% dataset increments. Because F-score is a combination of precision and recall, the same distributions can be seen as for those metrics. K-NN has the highest recall value, which means it was the most accurate.

1) 5-fold cross-validation:
   Fig. 12 presents 5-fold cross-validation performance metrics for the three ML models. Once more, K-NN outperformed the others in accuracy at 93.20%, and similarly had the best recall and F-score performance. In second place is SVM, and in third place is LR.
2) 10-fold cross-validation:
   Fig. 12 shows 10-fold cross-validation performance metrics for the three ML models. Even in the 10-fold case, K-NN performs the best with 96.29% accuracy, similarly high recall and F-score values. One can see that the recall is particularly high, which implies that the model is good at detecting malicious traffic. The LR model was outperformed by SVM in 5-fold cross-validation; however, LR in 10-fold cross-validation performed better than SVM.

While experimenting with a 0-byte payload using two PCs, it was noticed that a discrepancy in Wireshark is found, which is detailed in Table 14, and observation notes are added. During the emulation, six different ESP units were tried. Each generated different numbers of packets per second. The more control flow statements (loops or conditions) are used in the code, the fewer the packets it generates per minute i.e., the more the code written, the greater the computational process required.

It is worth mentioning that for normal ICMP request packets, a one-second delay is maintained as an interval between sending consecutive packets. In one of the experiments, this was exploited by ignoring this delay and sending an ICMP request packet as soon as the reply arrived. Waiting for a reply usually takes a fraction of a second. This style of attack does not saturate the buffer of the target, but keeps it busy.

A solution to this kind of attack is measuring the time difference between the request and response. If the difference is around 1 second, then it is considered to be normal traffic. However, if it is a fraction of a second, then it is concluded to be malicious. This calculation should be taken over an average of multiple instances, because timing is not entirely predictable in such IoT networks.

When performing the ping flood, after some time, the pcap logs showed that there were no echo responses to the ping flood. This is interpreted as the memory buffer being full.

To profile the memory usage for the Python ML program, memory_profiler and Scalene were used [61], [62]. It was found that most of the memory usage is for importing libraries, such as pandas and sklearn. In contrast, handling the dataset and training and testing the ML models had a tiny impact on the memory usage. This can be observed in Fig. 13, where the ML function resides below the edge of the maximum memory usage at 120 MB. Fig. 13 was produced for LR prediction using the memory_profiler library. This small memory usage for the ML function is to be expected because only 1000 samples/records were used. However, if a larger dataset were used, the memory usage would increase accordingly.

This framework also measured the amount of time for feature extraction from the Zeek tool until the production of CSV files, which is mentioned in Section IV-C (the spoofed experiment). This involved processing the pcap files and producing the CSV files. The time measurement, i.e., the execution time, was measured for both normal and malicious traffic at two specific times. The first is around 1 s and the second is near 60 s. Table 15 presents the detailed results of these experiments. The measurement was performed using the ''time'' command from the Linux OS. An executable shell script was passed as an argument, containing Zeek, zeek-cut, awk, and sed commands.

Fig. 14 presents the execution time or time consumption for Zeek and other tools up to the production of CSV files, for both normal and malicious traffic. One can observe that the malicious traffic required more time than the normal traffic. The normal traffic in both instances resulted in similar execution times, even after an increase in traffic.

1) COMPARATIVE ANALYSIS

The research in the literature applies a normal dataset that contains various protocols to train a model to detect ping flood attacks; however, our normal dataset only contains normal ping traffic. Furthermore, the normal dataset is unbiased in terms of uniqueness, i.e., no observation is duplicated.

Although researchers have applied numerous stateless (packet header information) and stateful (flow information) features, our work employs the optimal and base minimum of flow features based on the experience of generating ping flood packets from scratch. The two features chosen offer efficient training and testing times.

Table 16 presents the differences in accuracy between this work and others. The work [20] has a publicly available dataset that contains many attacks. One of the files in the dataset [35] has a single ping flood attack pcap file,

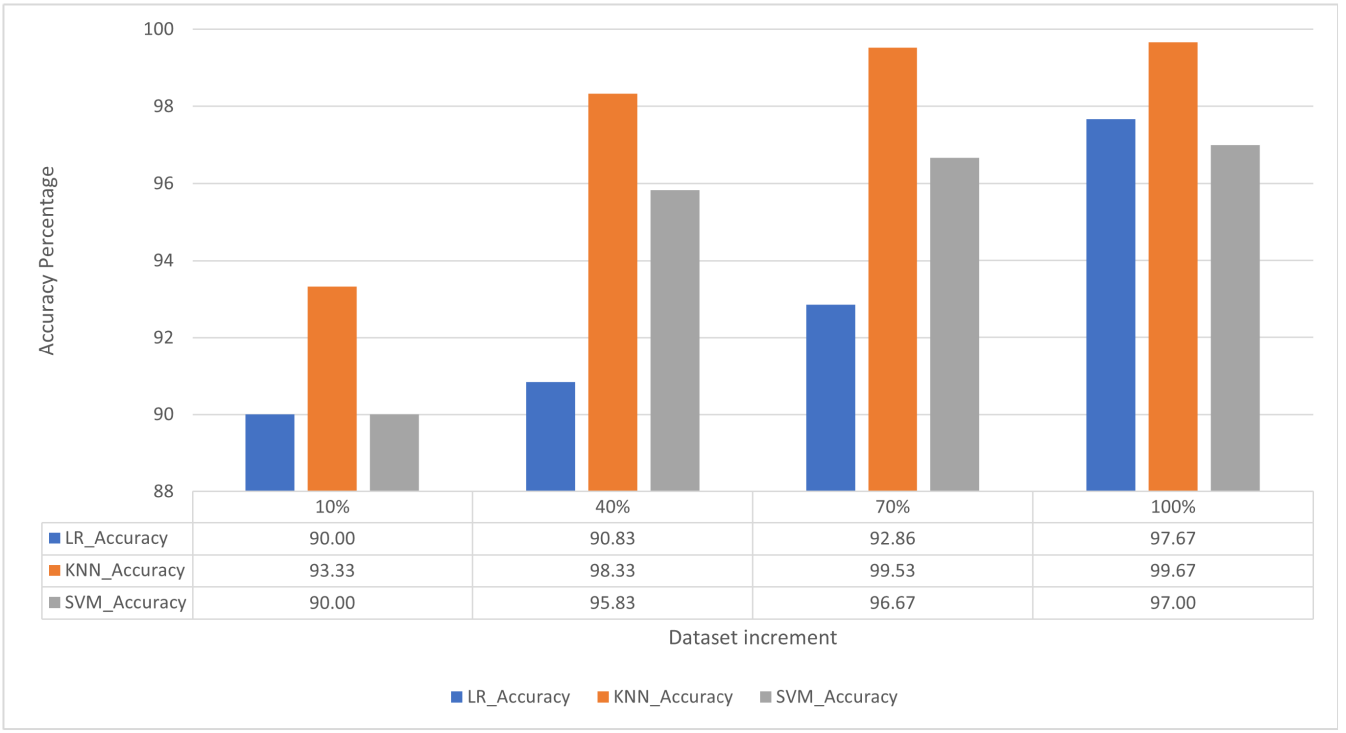

FIGURE 8. Accuracy of three ML models across data increments for split validation.

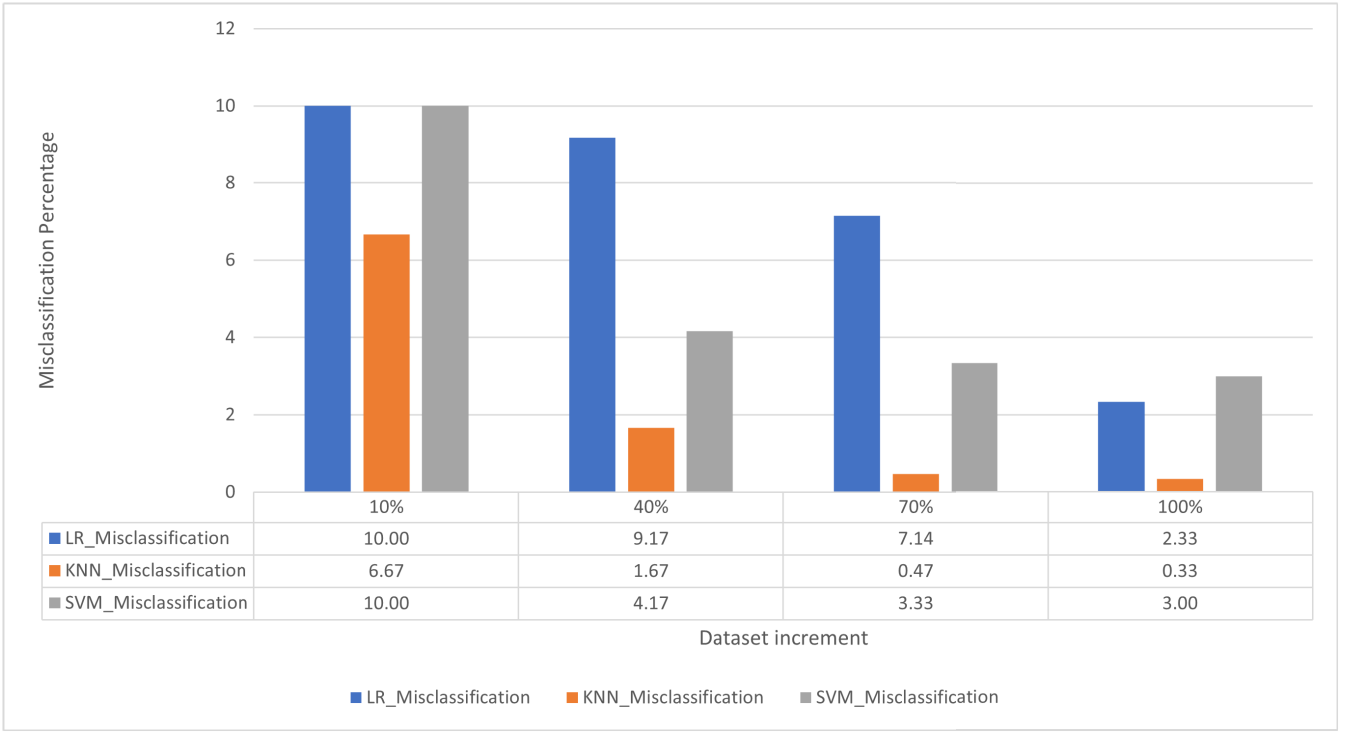

FIGURE 9. Misclassification of three ML models across data increments for split validation.

named "DDoS ICMP Flood Attacks.pcap" (their dataset was generated from hping3, implying a PC was used). After downloading and applying INIDF feature extraction, one record (time window) was obtained from the extraction,

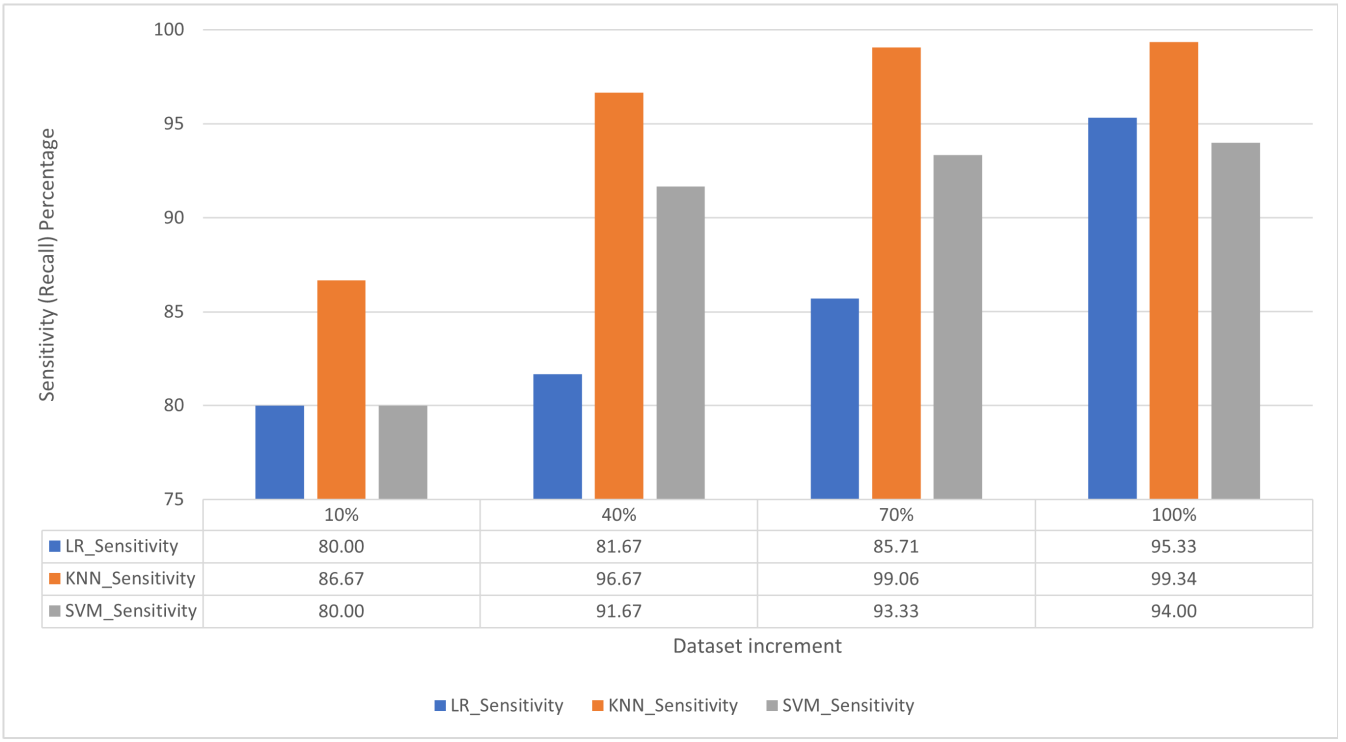

FIGURE 10. Recall of three ML models across data increments for split validation.

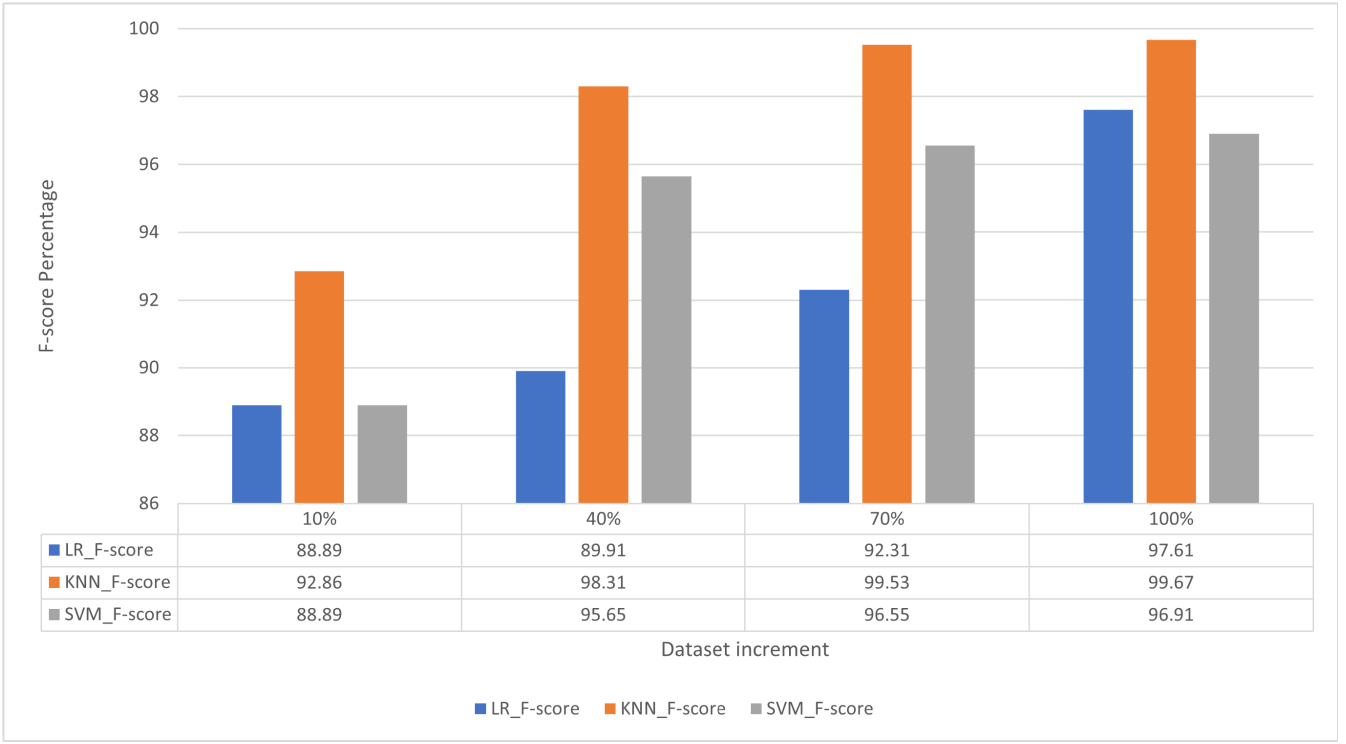

FIGURE 11. F-score of three ML models across data increments for split validation.

the results of which are as follows: the duration was 403.675 seconds and the number of echo requests was 2 411 190 packets. Subsequently, ML was applied, producing the following results: LR=1, K-NN=1, and SVM=0. In this

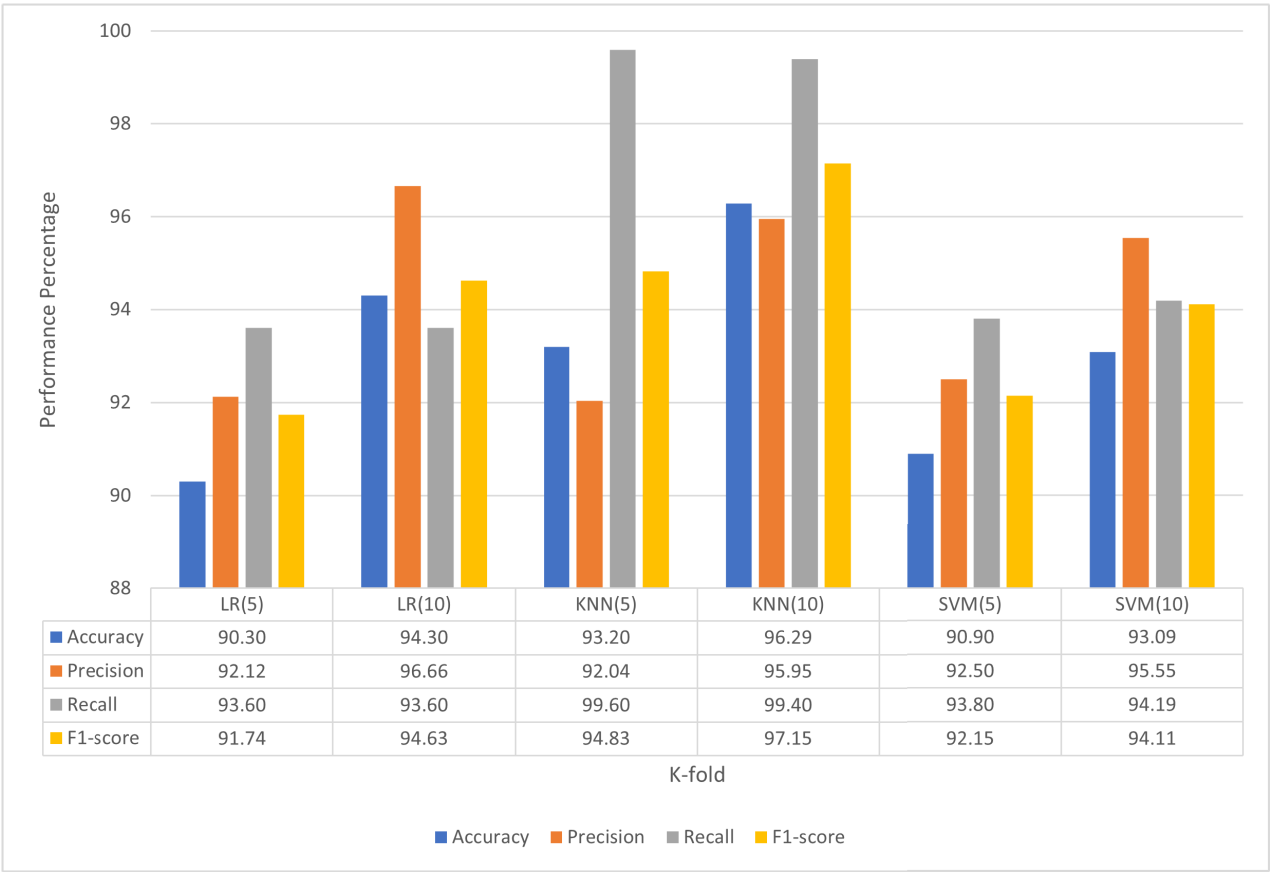

FIGURE 12. 5- and 10-fold cross-validation performance metrics for the three ML models.

TABLE 14. Performance metrics.

No	PC sender	Direction	PC receiver
1	Sent ICMP request packet with 0 data size. Wireshark shows 42 bytes have been sent.	Request	Wireshark shows a received ICMP request packet with 60 bytes.
2	Wireshark shows a received ICMP request packet with 60 bytes.	Response	Sent ICMP reply packet with 0 data size. Wireshark shows 42 bytes have been sent.
Observation notes			
On the PC receiver side, the packet showed 0 data; however, the 18 bytes are added in the data-link layer "padding field." Padding is implemented in an ICMP packet if the data size is less than 18 bytes. This means that the padding is done after registering the packet in Wireshark. To be clear, every OS has its own implementation; therefore, this information may differ in other OSs or OS versions. The minimum frame size of Ethernet v2 is 64 bytes. There are 4 bytes missing, which are for the frame check sequence (FRC). Wireshark does not show the frame checksum, which is why it shows 60 bytes instead of 64 bytes, as illustrated in Figs. 15 and 16. The payload of an ICMP echo request/response never changes. It is deduced that if a payload of size 1000 is sent, 1000 is expected to return with the same data.			

TABLE 15. Time measurement for using Zeek tool up to CSV files.

Normal			Malicious		
Duration of ICMP requests (seconds)	Number of ICMP requests	Zeek and other tools execution time (seconds)	Duration of ICMP requests (seconds)	Number of ICMP requests	Zeek and other tools execution time (seconds)
1.00066 s	2	0.2946 s	1.80368 s	1327	0.318s
59.083674 s	60	0.2964 s	60.8764 s	39 639	1.973s

case, 1 is the desired outcome, because the file contains a ping flood and "1" means malicious. Even though the dataset is larger than the dataset generated in this work, in terms of echo requests in a time window, K-NN, the best model achieved in this study, was able to distinguish malicious traffic. This result demonstrates the abilities of INIDF.

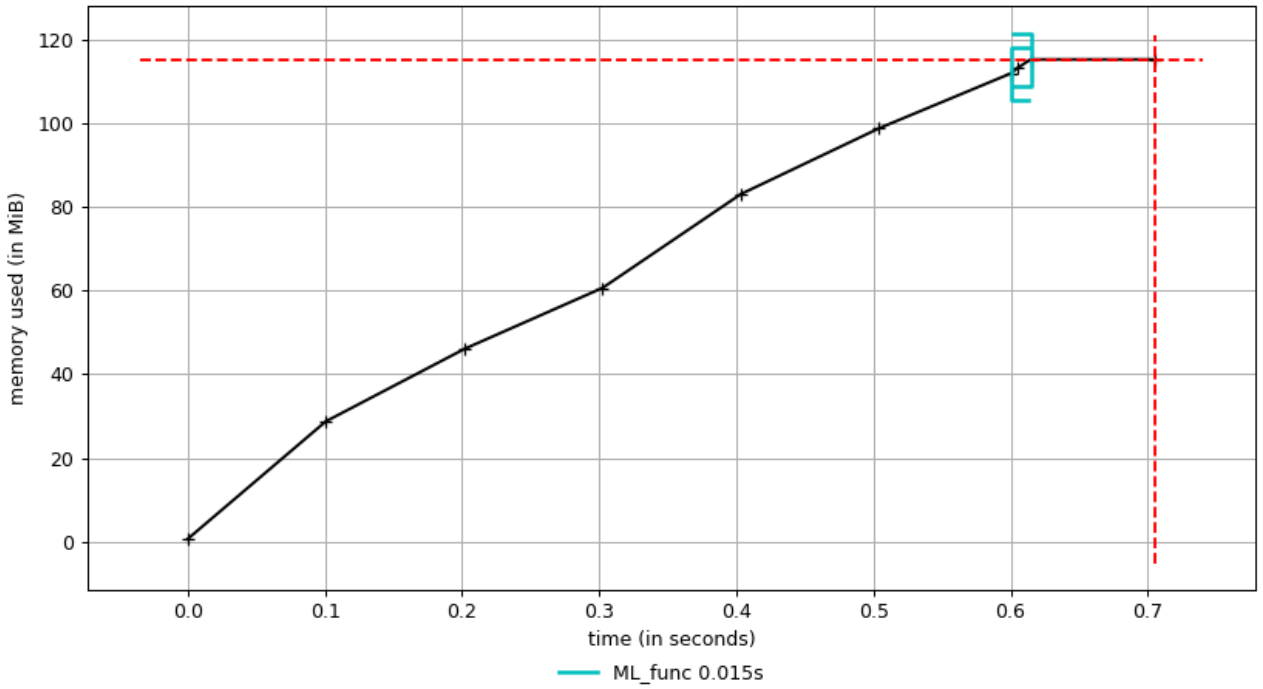

FIGURE 13. Memory usage for Python program, ML_function, which performs dataset handling, training, and testing. The image was produced using the memory_profiler library.

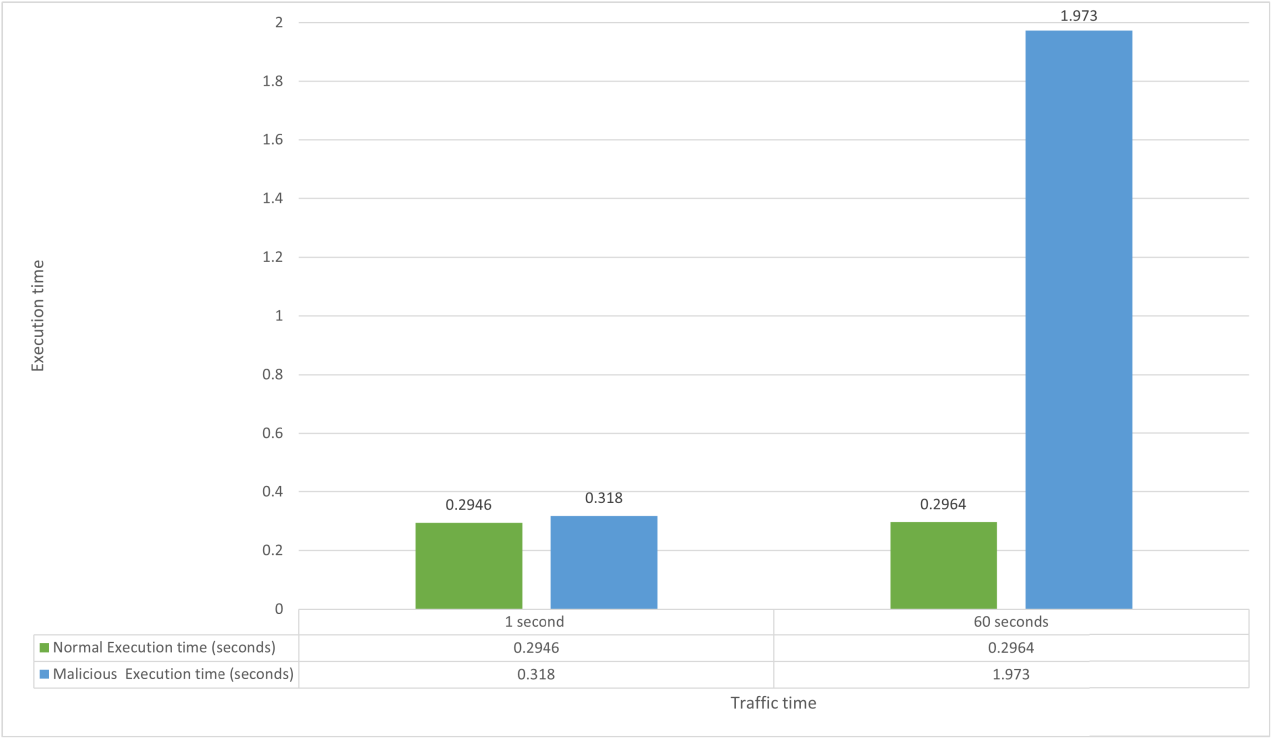

FIGURE 14. Execution time for Zeek and other tools up until CSV files, i.e., feature extraction.

2) LIMITATIONS AND FUTURE WORK

The framework is able to detect if the time window is malicious or normal; however, because attackers use spoofed IP addresses, differentiating between legitimate and malicious IP addresses is still an unattainable goal. As mentioned above, inspecting individual packets is futile,

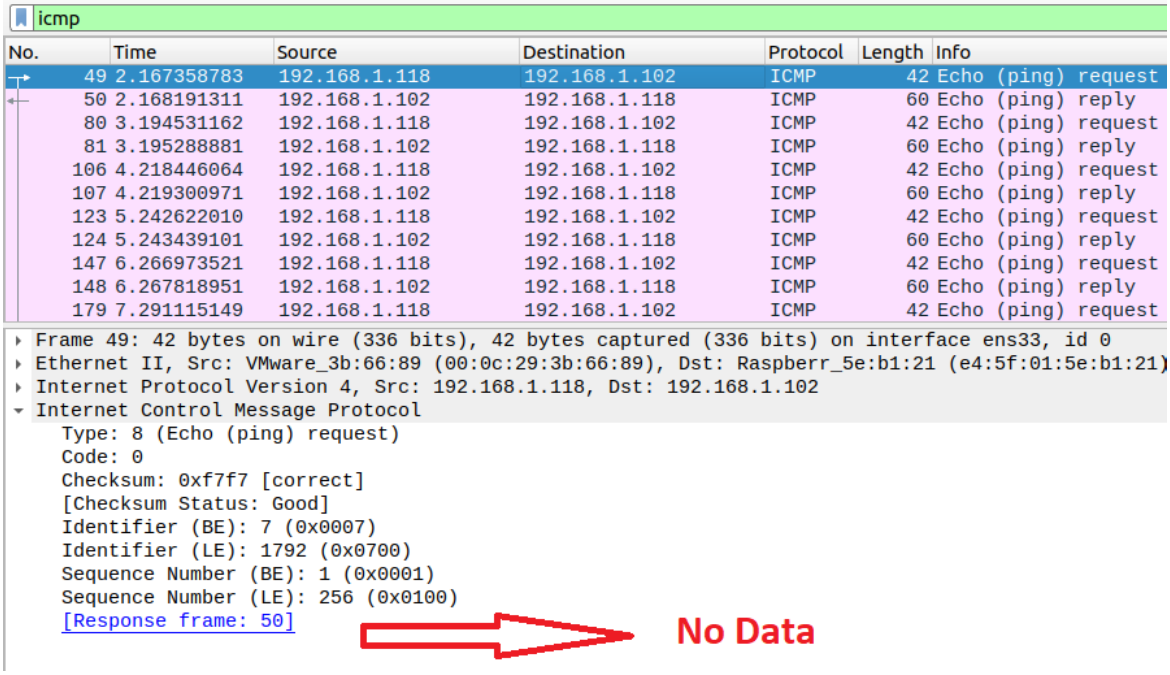

FIGURE 15. Snapshot of Wireshark portraying an echo request with zero data size.

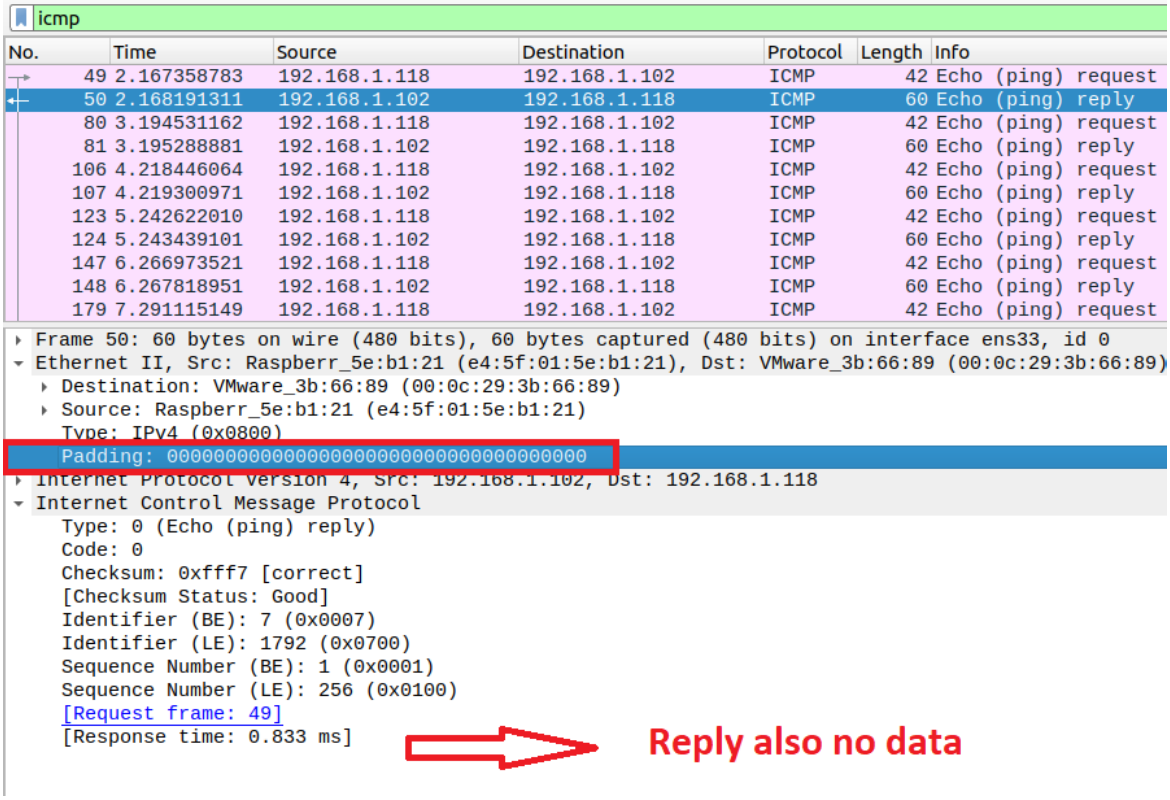

FIGURE 16. Snapshot of Wireshark portraying an echo response with zero data size but with an increased frame length to 60 bytes because of the added padding field.

so future solutions should revolve around flow information. Furthermore, this work does not address ICMP fragmentation attacks, also other ML and DL algorithms can be explored to see which can perform best in accuracy as well as time

TABLE 16. Comparison table of our work with other research.

Research	Accuracy
Chen et al.(2020)	97%
Lee et al. (2022)	99%
Ferrag et al. (2022)	99.99%
Doshi et al. (2018)	99%
Stiawan et al. (2021)	99.94%
Gupta et al. (2022)	97%–99%
Our INIDF	97%–99.67%

TABLE 17. Nomenclature for key terms.

Name	Definition
DDoS	Distributed Denial of Service
DoS	Denial of Service
echo request/ping	Is one of the functionalities of ICMP that uses a request to test the connectivity and measure the latency.
ICMP	Internet Control Messaging Protocol
IDS	Intrusion Detection System are software that detect malicious or abnormal traffic to a system.
INIDF	IoT network intrusion detection framework
IoT	Internet of Things
ML	Machine Learning uses historical data to create a model for predicting the outcome of new data.
pcap	Packet capture files that contain network traffic.
Zeek	Is a tool for traffic analysis.

consumption. These matters are left as potential avenues for future extension of the framework.

VI. CONCLUSION

This work has investigated current research for a ping flood attack dataset generated from an embedded device. However, all datasets reviewed were either generated from a tool originating from a higher processing speed device or had IPs manipulated to appear as if they came from an embedded device. This work adds to the body of knowledge with regard to embedded device attack volumes and offers a new methodological approach. The framework developed two datasets: normal ping traffic and a malicious ping flood attack. Since we developed the ping packets, we know their characteristics; therefore, we extracted the optimal and base minimum of features from the ping packets. We then trained and tested three ML algorithms. The models produced were evaluated by calculating the confusion matrix and its derivatives. The execution times for feature extraction and ML were measured, and the difference in bandwidth between different classes of devices was exhibited. The memory usage of the ML process was also analyzed. The best model achieved by the framework was the K-NN model, with an accuracy of 99.67%.

The most important future extension of this work would be to develop a method to identify legitimate ping request users. Despite detecting malicious ping flood attacks in a time window, we were not able to distinguish between legitimate users and attackers, due to the fact that malicious parties use spoofed IP addresses to conceal themselves. Additionally, further ML and DL algorithms could be explored.

NOMENCLATURE

See Table 17.

REFERENCES

[1] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An in-depth analysis of IoT security requirements, challenges, and their countermeasures via software-defined security," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10250–10276, Oct. 2020.

[2] K. Rose, S. Eldridge, and L. Chapin, "The Internet of Things: An overview," *Internet Soc. (ISOC)*, vol. 80, pp. 1–50, Oct. 2015.

[3] B. Krebs. (2018). *Study: Attack on KrebsOnSecurity Cost IoT Device Owners $323K*. Accessed: Mar. 20, 2023. [Online]. Available: https://krebsonsecurity.com/2018/05/study-attack-on-krebsonsecurity-cost-iot-device-owners-323k/

[4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, and M. Kallitsis, "Understanding the Mirai botnet," in *Proc. 26th USENIX Secur. Symp. (USENIX Security)*, 2017, pp. 1093–1110.

[5] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[6] S. Selvarajan, M. Shaik, S. Ameerjohn, and S. Kannan, "Mining of intrusion attack in SCADA network using clustering and genetically seeded flora-based optimal classification algorithm," *IET Inf. Secur.*, vol. 14, no. 1, pp. 1–11, Jan. 2020.

[7] S. Shitharth, N. Satheesh, B. P. Kumar, and K. Sangeetha, "IDS detection based on optimization based on WI-CS and GNN algorithm in SCADA network," in *Architectural Wireless Networks Solutions and Security Issues*. Singapore: Springer, 2021, pp. 247–265.

[8] S. Shitharth, P. R. Kshirsagar, P. K. Balachandran, K. H. Alyoubi, and A. O. Khadidos, "An innovative perceptual pigeon galvanized optimization (PPGO) based likelihood Naïve Bayes (LNB) classification approach for network intrusion detection system," *IEEE Access*, vol. 10, pp. 46424–46441, 2022.

[9] N. Eddermoug, A. Mansour, M. Sadik, E. Sabir, and M. Azmi, "*Klm*-PPSA v. 1.1: Machine learning-augmented profiling and preventing security attacks in cloud environments," *Ann. Telecommun.*, pp. 1–27, Jul. 2023.

[10] *Internet Control Message Protocol*, document RFC 792, Mar. 2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc792

[11] *Zeek Documentation*. Accessed: Mar. 20, 2023. [Online]. Available: https://docs.zeek.org/en/master/

[12] *Zeek Github Repository*. Accessed: Mar. 20, 2023. [Online]. Available: https://github.com/zeek/zeek

[13] *LwIP Github Repository*. Accessed: Mar. 20, 2023. [Online]. Available: https://github.com/lwip-tcpip/lwip

[14] *LWIP Wiki*. Accessed: Mar. 20, 2023. [Online]. Available: https://lwip.fandom.com/wiki/LwIP_Wiki

[15] *LWIP—A Lightweight TCP/IP Stack—Summary*. Accessed: Mar. 20, 2023. [Online]. Available: https://savannah.nongnu.org/projects/lwip/

[16] S.-H. Lee, Y.-L. Shiue, C.-H. Cheng, Y.-H. Li, and Y.-F. Huang, "Detection and prevention of DDoS attacks on the IoT," *Appl. Sci.*, vol. 12, no. 23, p. 12407, Dec. 2022.

[17] M. Arshi, M. Nasreen, and K. Madhavi, "A survey of DDoS attacks using machine learning techniques," in *Proc. E3S Web Conf.*, vol. 184, 2020, p. 01052.

[18] N. S. Vishnu, R. Singh Batth, and G. Singh, "Denial of service: Types, techniques, defence mechanisms and safe guards," in *Proc. Int. Conf. Comput. Intell. Knowl. Economy (ICCIKE)*, Dec. 2019, pp. 695–700.
[19] L. Chen, S.-E. Weng, C.-J. Peng, H.-H. Shuai, and W.-H. Cheng, "ZYELL-NCTU NetTraffic-1.0: A large-scale dataset for real-world network anomaly detection," in *Proc. IEEE Int. Conf. Consum. Electron.-Taiwan (ICCE-TW)*, Sep. 2021, pp. 1–2.
[20] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning," *IEEE Access*, vol. 10, pp. 40281–40306, 2022.
[21] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 29–35.
[22] D. Stiawan, M. E. Suryani, Susanto, M. Y. Idris, M. N. Aldalaien, N. Alsharif, and R. Budiarto, "Ping flood attack pattern recognition using a K-means algorithm in an Internet of Things (IoT) network," *IEEE Access*, vol. 9, pp. 116475–116484, 2021.
[23] B. B. Gupta, P. Chaudhary, X. Chang, and N. Nedjah, "Smart defense against distributed denial of service attack in IoT networks using supervised learning classifiers," *Comput. Electr. Eng.*, vol. 98, Mar. 2022, Art. no. 107726.
[24] B. Bouyeddou, F. Harrou, Y. Sun, and B. Kadri, "Detection of smurf flooding attacks using Kullback–Leibler-based scheme," in *Proc. 4th Int. Conf. Comput. Technol. Appl. (ICCTA)*, May 2018, pp. 11–15.
[25] T. Subbulakshmi, K. BalaKrishnan, S. M. Shalinie, D. AnandKumar, V. GanapathiSubramanian, and K. Kannathal, "Detection of DDoS attacks using enhanced support vector machines with real time generated dataset," in *Proc. 3rd Int. Conf. Adv. Comput.*, Dec. 2011, pp. 17–22.
[26] Y. Zhang, J. Xu, Z. Wang, R. Geng, K.-K. R. Choo, J. A. Pérez-Díaz, and D. Zhu, "Efficient and intelligent attack detection in software defined IoT networks," in *Proc. IEEE Int. Conf. Embedded Softw. Syst. (ICESS)*, Jul. 2020, pp. 1–9.
[27] S. Wankhede and D. Kshirsagar, "DoS attack detection using machine learning and neural network," in *Proc. 4th Int. Conf. Comput. Commun. Control Autom. (ICCUBEA)*, Aug. 2018, pp. 1–5.
[28] J. David and C. Thomas, "Efficient DDoS flood attack detection using dynamic thresholding on flow-based network traffic," *Comput. Secur.*, vol. 82, pp. 284–295, May 2019.
[29] O. Hannache and M. C. Batouche, "Neural network-based approach for detection and mitigation of DDoS attacks in SDN environments," *Int. J. Inf. Secur. Privacy*, vol. 14, no. 3, pp. 50–71, Jul. 2020.
[30] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A DDoS attack detection method based on SVM in software defined network," *Secur. Commun. Netw.*, vol. 2018, pp. 1–8, Jun. 2018.
[31] M. Aslam, D. Ye, A. Tariq, M. Asad, M. Hanif, D. Ndzi, S. A. Chelloug, M. A. Elaziz, M. A. A. Al-Qaness, and S. F. Jilani, "Adaptive machine learning based distributed denial-of-services attacks detection and mitigation system for SDN-enabled IoT," *Sensors*, vol. 22, no. 7, p. 2697, Mar. 2022.
[32] J. A. Pérez-Díaz, I. A. Valdovinos, K. R. Choo, and D. Zhu, "A flexible SDN-based architecture for identifying and mitigating low-rate DDoS attacks using machine learning," *IEEE Access*, vol. 8, pp. 155859–155872, 2020.
[33] Y.-W. Chen, J.-P. Sheu, Y.-C. Kuo, and N. Van Cuong, "Design and implementation of IoT DDoS attacks detection system based on machine learning," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, 2020, pp. 122–127.
[34] *Design and Implementation of IoT DDoS Attacks Detection System Based on Machine Learning (Dataset)*. Accessed: Mar. 20, 2023. [Online]. Available: http://smartcity.cs.nthu.edu.tw/dataset/download.php
[35] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke. *EDGE-IIOTSET: A New Comprehensive Realistic Cyber Security Dataset of IOT and IIOT Applications: Centralized and Federated Learning*, Accessed: Mar. 20, 2023. [Online]. Available: http://ieee-dataport.org/8939
[36] *Ping Flood Attack Pattern Recognition on Internet of Things Network Dataset*. Accessed: Jun. 26, 2023. [Online]. Available: https://zenodo.org/record/4436208
[37] *1999 DARPA Intrusion Detection Evaluation Dataset*. Accessed: Jun. 26, 2023. [Online]. Available: http://ieee-dataport.org/8939
[38] *Sonoff Tech*. Jul. 16, 2023. [Online]. Available: https://sonoff.tech/
[39] *Flask's Documentation*. Accessed: Jul. 16, 2023. [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/
[40] *Dumpcap Manual*. Accessed: Jul. 16, 2023. [Online]. Available: https://www.wireshark.org/docs/man-pages/dumpcap.html
[41] O. Almorabea, T. Khanzada, M. Aslam, F. Hendi, and A. Almorabea. *IoT Emulated Dataset for ICMP/Ping Normal and Malicious Traffic*. Accessed: Jul. 4, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8111635
[42] *Code for Emulation of ICMP/Ping Normal and Malicious Traffic by using ESP-01s*. Accessed: Apr. 7, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8112222
[43] *How Does Ping Really Work George Mays*. Accessed: Jun. 26, 2023. [Online]. Available: http://hosteddocs.ittoolbox.com/WP_Mays_Ping.pdf
[44] M. Gogoi and S. Mishra, "Detecting DDoS attack using snort," Res. Gate, Tech. Rep., 2018.
[45] H. Lin and P. Wang, "Implementation of an SDN-based security defense mechanism against DDoS attacks," in *Proc. Joint Int. Conf. Econ. Manage. Eng. (ICEME) Int. Conf. Econ. Bus. Manage. (EBM)*, 2016.
[46] *The GNU Awk User's Guide*. Accessed: Mar. 20, 2023. [Online]. Available: https://www.gnu.org/software/gawk/manual/gawk.html
[47] *Awk Guide*. Accessed: Mar. 20, 2023. [Online]. Available: https://www.grymoire.com/Unix/Awk.html
[48] *GNU AWK*. Accessed: Mar. 20, 2023. [Online]. Available: https://learnbyexample.github.io/learn_gnuawk/cover.html
[49] *Sed, A Stream Editor*. Accessed: Mar. 20, 2023. [Online]. Available: https://www.gnu.org/software/sed/manual/sed.html
[50] *Sed Linux*. Accessed: Mar. 20, 2023. [Online]. Available: https://linux.die.net/man/1/sed
[51] *MinMaxScaler Formula*. Accessed: Jun. 26, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
[52] M. V. K. Jewani, P. E. Ajmire, and M. G. N. Brijwani, "Using machine learning techniques to detect distributed denial of service attacks," *IJSRST*, vol. 9, no. 6, pp. 265–271, Sep./Oct. 2021.
[53] N. K. Rangaraju, S. B. Sriramoju, and S. Sarma, "A study on machine learning techniques towards the detection of distributed denial of service attacks," *Int. J. Pure Appl. Math.*, vol. 120, no. 6, pp. 7407–7423, 2018.
[54] M. A. Rahman, A. T. Asyhari, L. S. Leong, G. B. Satrya, M. H. Tao, and M. F. Zolkipli, "Scalable machine learning-based intrusion detection system for IoT-enabled smart cities," *Sustain. Cities Soc.*, vol. 61, Oct. 2020, Art. no. 102324.
[55] *SVM Formula*. Accessed: Jun. 26, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/SVM.html
[56] T. Saranya, S. Sridevi, C. Deisy, T. D. Chung, and M. A. Khan, "Performance analysis of machine learning algorithms in intrusion detection system: A review," *Proc. Comput. Sci.*, vol. 171, pp. 1251–1260, Jan. 2020.
[57] *LR Formula*. Accessed: Jun. 26, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#-logistic-regression
[58] U. Islam, A. Muhammad, R. Mansoor, M. S. Hossain, I. Ahmad, E. T. Eldin, J. A. Khan, A. U. Rehman, and M. Shafiq, "Detection of distributed denial of service (DDoS) attacks in IoT based monitoring system of banking sector using machine learning models," *Sustainability*, vol. 14, no. 14, p. 8374, Jul. 2022.
[59] *K-NN Formula*. Accessed: Jun. 26, 2023. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html
[60] *Fragmentation Flags Definition*. Accessed: Mar. 20, 2023. [Online]. Available: http://www.infroref.org/i3450flagsip.htm
[61] *Memory Profiler*. Accessed: Jul. 16, 2023. [Online]. Available: https://pypi.org/project/memory-profiler/
[62] *Scalene*. Accessed: Jul. 16, 2023. [Online]. Available: https://pypi.org/project/scalene/0.9.15/

OMAR MOHAMMED ALMORABEA received the bachelor's degree in information systems from the Faculty of Computing and Information Technology, King Abdulaziz University, where he is currently pursuing the master's degree with the Information Systems Department. His research interests include web development, networks, cybersecurity, the IoT, and robotics (mechatronics).

TARIQ JAMIL SAIFULLAH KHANZADA received the B.E. degree in CS and the Master of Engineering degree in CSN from Mehran UET, Jamshoro, Pakistan, in 1999 and 2003, respectively, and the Ph.D. degree in wireless information and communication engineering from the OVGU University of Magdeburg, Germany. He has been a Professor with the Department of Information Systems, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia, since September 2019. He is on leave (from September 2019) from the post of a Professor (BPS-21) with the Department of Computer Science and Engineering, Mehran UET, Jamshoro, where he has been a Professor, since October 2016. Previously, he was with the Department of Computer Science and Engineering as an Associate Professor, an Assistant Professor, and a Lecturer, in 2011, 2004, and 2001, respectively. He has presented his research in more than a dozen countries, including Germany, France, USA, Canada, The Netherlands, Italy, and Czech Republic. He has also achieved more than 15 meritorious grants and awards from various domestic and international research agencies (such as the Education Board, HEC, and DFG) during his academic and professional carrier. He is the author of a book and several international book chapters in various domains, namely IT, WSN, and DL, published by renowned publishers, in 2016, 2022, and 2023. He has more than 60 international conference and journal publications, including high impact factor publications. His research interests include wireless indoor positioning systems, signal processing for digital and wireless communications systems, and AI machine learning and data sciences applications.

MUHAMMAD AHTISHAM ASLAM received the master's degree in computer science from Hamdard University, Lahore, Pakistan, in 2002, and the Ph.D. degree from the University of Leipzig, Leipzig, Germany, in 2007. From 2003 to 2007, he was a Research Assistant with the University of Leipzig, Germany. He was a Senior Staff Researcher with the Artificial Intelligence Laboratory, Knowledge Technology Cluster, Malaysian Institute of Microelectronic Systems (MIMOS), Kuala Lumpur, Malaysia. He was an Assistant Professor with the COMSATS Institute of Information Technology, Lahore, and as an Associate Professor with the Department of Information Systems, King Abdulaziz University (KAU), Jeddah, Saudi Arabia. Currently, he is with FOKUS, Fraunhofer, Berlin, Germany, as a Researcher and as an Adjunct Full Professor with the Gujranwala Institute of Future Technologies (shortly GIFT University). His research interests include the semantic web, linked open data, data science, and social networks. He was a recipient of the 13th All Pakistan Dr. Abdul Qadir Khan Research Laboratories Software Competition Award, in 2002, the Partial Support Scholarship for Ph.D. Studies Abroad from the Higher Education Commission (HEC) of Pakistan, the Best Researcher Award from the Department of Information Systems, Faculty of Computing and Information Technology (FCIT), KAU (2016–2017), and the Best Session Research Paper Awards at the Eighth COMSATS Workshop on Research in Computing (CWRC 08), Wah Cantt, Pakistan.

FATHEAH AHMAD HENDI received the Ph.D. degree in pure mathematics from the Faculty of Science, King Abdulaziz University, Saudi Arabia. She has published books on the numerical solution of partial differential equations, and more than 50 articles on the numerical solution of equations. Her research interests include the numerical solution of integral equations, partial differential equations, and ordinary differential equations.

AHMAD MOHAMMED ALMORABEA is currently pursuing the Ph.D. degree with King Abdulaziz University. He is also a senior consultant in offensive security. Previously an Incident Responder, he is simultaneously attaining the Ph.D. degree. His research interest includes information security, especially in cryptography and its algorithms along with threat intelligence. Additionally, he has developed many mechanisms for analyzing intelligence feeds. He has also developed numerous open-source tools for the information security community and has discovered many vulnerabilities in major software: in particular, he has written exploit codes as proofs of concept for various vulnerabilities in Linux and Microsoft Windows. He has been a speaker at previous and upcoming black hat conferences.