http://dx.doi.org/10.21622/ACE.2026.06.1.1834

DESIGN AND EVALUATION OF A SOFTWARE-DEFINED NETWORKING INTRUSION DETECTION AND PREVENTION SYSTEM USING ENSEMBLE MACHINE LEARNING AND HYBRID FEATURE SELECTION

Olumhense Benedict Adoghe[1], Francis Emmanuel Kibuebu[1], Ejemen Comfort Ikpotokin[2], Emmanuel Dominic Ephraim[3]

[1]Achievers University, Department of Electrical and Information Technology Engineering, Owo, Ondo State

[2]Auchi Polytechnic University, Basic Science Department, Auchi, Edo State, Nigeria

[3]Achievers University, Department of Biomedical Engineering, Owo, Ondo State, Nigeria

Emails: {adoghe.ob@achievers.edu.ng, kibuebuemmanuelf@gmail.com, comfortikpotokin@gmail.com, ephraim.ed@achievers.edu.ng}

Received 29 November 2025 - Accepted 23 January 2026 - Published 22 April 2026

ABSTRACT

Cyber threats in today's software-defined network infrastructures are growing at an exponential rate; thus, sophisticated, adaptive security measures are required. In order to enhance detection accuracy and computational efficiency, this study introduces a Network Intrusion Detection and Prevention System (NIDPS) that is ensemble-based and designed for Software-Defined Networking (SDN). The NIDPS makes use of a hybrid feature selection technique. As part of its soft-voting ensemble architecture, the system incorporates XGBoost, Decision Tree, and Support Vector Machine, three machine learning classifiers. To tackle the difficulties of dealing with high-dimensional network traffic data, feature selection is optimised using Correlation-Based methods and Recursive Feature Elimination (RFE). Datasets obtained from simulated Denial-of-Service (DoS) attacks were used to evaluate the model, which was constructed and tested in a virtualised, emulated SDN environment using the SEED Internet Emulator. All of the model variations demonstrated near-perfect detection ability (up to 100% accuracy) in the experiments, with the fastest predictions coming from RFE-enhanced models. The system is well-suited for implementation in actual programmable network settings due to its real-time alerting and preventive features. The results of this study show that an effective and scalable method for intrusion detection in SDNs may be achieved by combining ensemble learning with intelligent feature selection.

Keywords: *Network Intrusion Detection System (NIDS), Ensemble Machine Learning, Hybrid Feature Selection, Cybersecurity, Denial-of-Service (DoS) Attack Simulation*

INTRODUCTION

The radical change in data movement and network activity created by the groundbreaking influence of digital technology, cloud computing, and the IoT on modern communication infrastructures has led to a dramatic increase in data transfer and network activity [1, 2]. Due to the increase in cyberattacks, the number and destructive power of cyberattacks are rising, and because of these developments, novel issues in cybersecurity emerge [3]. Recent statistics show that network attacks will constitute a significant portion of the global damages of cybercrime that will wipe out over $10.5 trillion annually by 2025 [4, 5]. Intrusion Detection Systems (IDS) have become an invaluable security measure in this setting due to the burden of threat identification, mitigation, and prevention in real-time [6, 7]. Conventional methods of IDS, like signature-based ones, are based on given

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

attack patterns, and many are not able to respond to zero-day attacks or changes in threat vectors [8]. The same cannot be said about anomaly-based detection, where ML and DL are employed to identify suspected network activity and allow more latitude to detect a new threat [9]. Notwithstanding these advancements, the current IDS can yet achieve the objectives to overcome the challenges facing them, such as computational inefficiency, class imbalance, large false-positive rates (FPR), and high-dimensional data [10]. Recently, hybrid feature selection and ensemble machine learning models have been popular as solutions to these shortcomings [11]. These methods integrate the strengths of many algorithms to enhance the detection accuracy, generalisability, and robustness. This article explores how better network intrusion detection can be achieved by using enhanced ensemble ML models along with hybrid feature selection methods to draw a comprehensive overview of current methods, challenges, and possible future directions of the topic. The online transformation of industries, smart cities, and critical infrastructure exposes networks to more attacks since a larger attack surface is created. The Cybersecurity Ventures 2024 Report shows that one cyberattack happens every eleven seconds on average. Ransomware, Distributed Denial-of-Service (DDoS), and Advanced Persistent Threats (APTs) are considered to be the most prevalent forms of cyberattacks [12, 13]. Besides this, IoTID20 and CICIDS2017 evidence indicates that emerging threats have complicated patterns and regularly succeed in evading conventional security controls [14]. The issues with conventional approaches to intrusion detection systems (IDS) are that they have low generalisability due to their reliance on static attack signatures, high false-positive rates (FPR) resulting in false alarms causing unwarranted operational costs, and cannot support high-dimensional data owing to their inability to process large volumes of network traffic efficiently [15].

The constraints underscore the significance of smart intrusion detection systems (IDS) that are adaptive and can scale according to the needs of the modern cybersecurity threats. With the development of deep learning and machine learning, today, intrusion detection systems can automatically generate features, detect abnormalities, and classify threats in real-time [16]. With known attacks, RF, XgBoost, and neural networks can also be successfully applied in the case of supervised learning, whereas unsupervised (and semi-supervised) algorithms such as Autoencoders and LSTM are particularly effective when it comes to zero-day attacks [17, 18]. Recent research has demonstrated that hybrid DL models, such as CNN-LSTM and GRU-based models, can be very effective at capturing a sequence and spatial pattern of network traffic, as demonstrated in recent studies [17]. For instance, the XGBoost-LSTM hybrid model was capable of performing better than traditional methods with a 99.7 percent detection rate on the CICIDS2017 dataset [19]. Similarly, Deep Reinforcement Learning (DRL) has been employed in advancing the skills of detection by modifying the evolving methods of attack [20].

The biases in the models exist due to the class imbalance in which benign samples of traffic far exceed the amount of attack samples, making the models biased [9]; redundancy in features and noise, worsening the performance of the model and causing longer training sequences, are issues of concern [18, 21] that the ML/DL-based IDS continue to face. The challenges require the incorporation of sophisticated feature selection methods in order to maximize the efficiency and accuracy of the model. One of the most important preprocessing methods in IDS is feature selection, whose main purpose is to minimize dimensionality, remove irrelevant features, and enhance detection efficiency [22]. Conventional feature selection techniques can fall into either of the following categories: Filter Methods (e.g., Pearson Correlation, Chi-square, ANOVA) - Fast yet do not consider interactions between features 213, Wrapper Methods (e.g., Recursive Feature Elimination, Genetic Algorithms) - Slow but model-sensitive 3, and Embedded Methods (e.g., Lasso Regression, Random Forest Importance) - between efficiency and performance [23].

Recent investigation suggests that hybrid feature selection can increase intrusion detection performance by combining various feature selection strategies and capitalising on their individual strengths. To take advantage of the synergies between different feature selection paradigms, hybrid feature selection combines many approaches, most commonly filter and wrapper methods [8], [15]. Before training the model, this study uses correlation-based filtering to remove features that are highly correlated or redundant.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

Then, it uses Recursive Feature Elimination (RFE) to find the features that are most discriminative. Improved computing economy without sacrificing classification accuracy is achieved using this hybrid technique [21]. As an illustration, MI-Boruta (Mutual Information + Boruta Algorithm) offered 99.9% accuracy on UNSW-NB15 with the combination of filter and wrapper, CFS-FPA (Correlation Feature Selection + Forest Penalized Attributes) had 0.004% false alarms on CICIDS2017, and the selection of features with XGBoost offered higher detection rates at lower training times of 27-63% [13], [24]. The advantages of these combined methods are not only the improvement of detection accuracy but also the guarantee of practical use in high-speed networks in real-time. Such methods as Bagging, Boosting, and Stacking of ensemble learning have become popular in IDS as they enable a combination of several weak classifiers into a formidable detector [14], [24]. Some of the strategies that are used as ensembles are Random Forest (RF) & XGBoost - Effective on binary and multi-class classification 9, AdaBoost and Gradient Boosting - Improve minority class detection in imbalanced datasets 3, and Stacked Ensemble Models (e.g., RF + XGBoost + MLP Meta-learner) - Achieve superior generalization [25]. Recent research findings have proven that hybrid feature selection stacked ensemble models are better than single classifiers. In the example, an RF-CatBoost-XGBoost model won 99.99% on NSL-KDD and CICIDS2017, and an AdaBoost-enhanced SVM lowered false negatives by half over classic SVM [26]. These results indicate the promise of ensemble-hybrid IDS models to overcome recent cybersecurity issues.

METHODOLOGY

This section of the research presents the approach adopted to conduct, design, develop, and implement an ensemble-based NIDPS within an SDN architecture.

STAGES OF PROJECT DEVELOPMENT

Here, the critical stages of the project execution are presented with a focus on the iterative process of design, development, implementation, and evaluation metrics to measure the efficiency of the system. Figure 1 below provides a stepwise methodology structure that is depicted in phases.

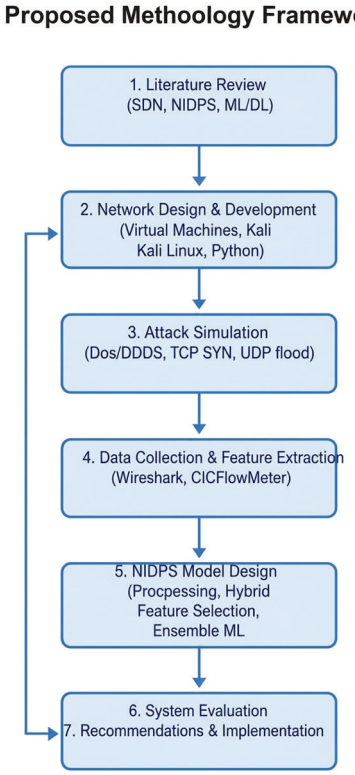

Figure 1: Proposed methodology framework

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

To ensure that the intrusion detection and prevention system is robust and effective, this study on Adopting Network Intrusion Detection by using the improved Ensemble Machine Learning Models with a Hybrid feature selection Model conforms to a process of steps. The initial one is to read the literature available on the topic at hand, which will provide you with an overview of such key technologies as Software-Defined Networking (SDN), NIDPS, and the past solutions. After this, a software-based network is designed and developed on virtual machines, and with the help of software such as Kali Linux, the network architecture is written and configured in Python. A virtual network is used to simulate Denial-of-Service (DoS) attacks, in which custom TCP SYN and UDP flood scripts are used to replicate the conditions of a real intrusion. During the data collection stage, Wireshark is used to capture network traffic of such attacks, and CICFlowMeter is used to extract features of the packet capture (.pcap) files and create machine learning-processable datasets. The second step is the development and design of an NIDPS model that takes into consideration the preprocessing of the obtained data, the use of hybrid feature selection algorithms to minimise dimensionality, and the governance of ensemble machine learning algorithms in order to increase detection accuracy. Lastly, the system is evaluated based on the performance in accordance with certain metrics to evaluate its efficiency, and the paper ends with recommendations for the implementation of the developed NIDPS system into practice, pointing to its practical importance and possible implications on the organizational cybersecurity practices.

DESIGN AND IMPLEMENTATION OF THE ENSEMBLE-BASED NIDPS

This part of the research paper presents the information about the design of the ensemble ML-based NIDPS that is integrated with an SDN architecture and tested on the performance measures. It starts with the network architecture design that illustrates the emulation of an advanced network, then the development and implementation of the NIDPS model in the isolated network. All the complex procedures that are entailed in this network design, the development of the NIDPS model, implementation, and evaluation are all reflected in this section. The proposed system has the overall network architecture as illustrated in Figure 2 below.

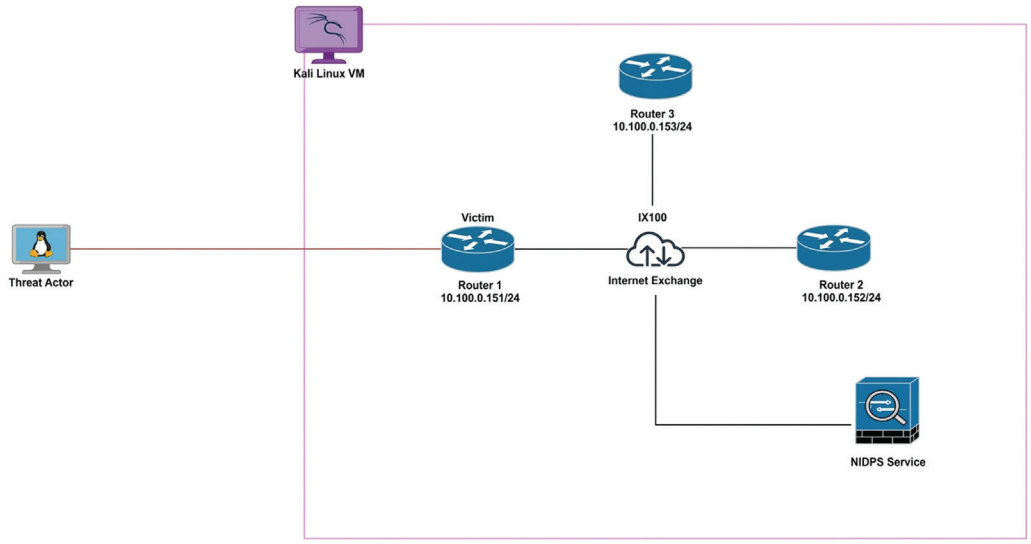

Figure 2: System Design and Architecture

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

VIRTUAL MACHINE (VM) SETUP AND CONFIGURATION

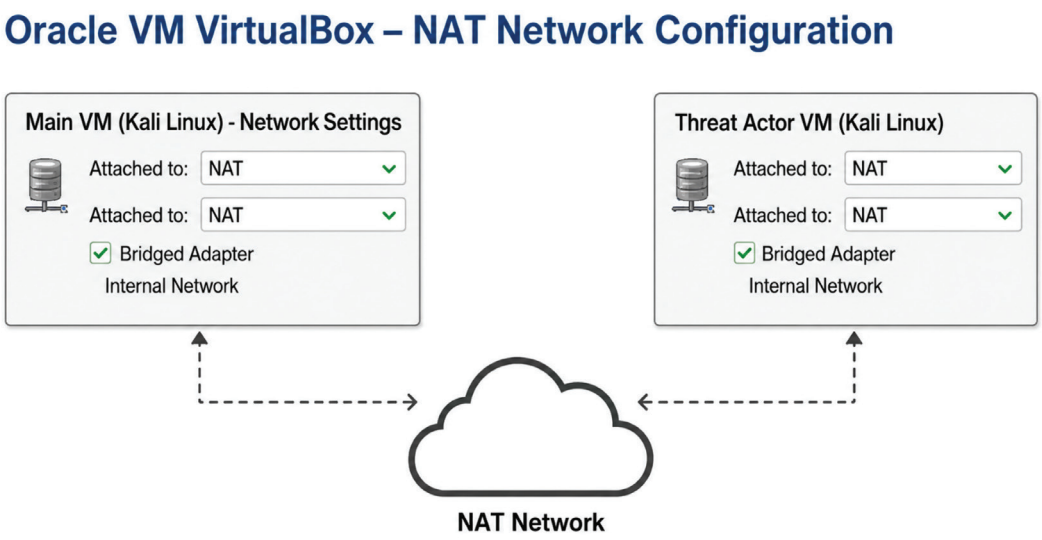

Figure 3: VM Setup and Configuration

In order to safely design an NIDPS, tested against simulated attacks, it was necessary to establish an isolated environment in which the methods of development and testing may be conducted. Therefore, Kali Linux OS was installed in the Oracle VM VirtualBox manager that was endowed with 4GB base memory, 2 processors, 80GB virtual space, and access to the internet was disabled to ensure that any accidental exposure of simulated threats to the internet did not occur. The details of this setup are indicated in Figure 3 below.

The second VM set up was performed as shown in Figure 4. This was the mechanism that the threat agent used in delivering DoS/DDoS attacks. Kali Linux OS was also installed in the VM, and it had 2GB of memory and 2 CPU processors. The threat actor-VM was then connected to the same network as the main VM, as shown in Figure 4, due to the configuration.

```
#!/usr/bin/env python3
# encoding: utf-8
from seedemu.layers import Base, Routing, Edge
from seedemu.services import Webservice
from seedemu.compiler import Docker
from seedemu.core import Emulator, Binding, Filter

# Function to create an autonomous system with a router connected to an IX
def setup_autonomous_system(base_layer, asn, router_name, ix_network):
    asys = base_layer.create_autonomous_system(asn)
    asys.createrouter(router_name).joinnetwork(ix_network)
    return asys

# Initialize the emulator and layers
emu = Emulator()
base = Base()
routing = Routing()
edge = Edge()

# Create an Internet Exchange
ix_number = 100
base.createInternetExchange(ix_number)

# Autonomous system numbers for the three ASes
as_numbers = [150, 351, 353]

# Setup autonomous systems
for asn in as_numbers:
    setup_autonomous_system(base, asn, f'router{asn - 150}', f'ix{ix_number}')

# Peering the ASes at Internet Exchange IX-100
for asn in as_numbers:
    edge.addaspeer(ix_number, asn)

# Add layers to the emulator and render
emu.addlayer(base)
emu.addlayer(routing)
emu.addlayer(edge)
emu.render()

# Compile the emulation environment
emu.compile(Docker(), './output')
```

Figure 4: Network Address Translation Configuration for both VMs.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

NETWORK DEVELOPMENT

This was followed by the development of an emulated network on the SEED Internet Emulator (IE) software tool at this level, after having an isolated environment. The SEED IE provides a wide range of programmability of the network with configurable classes and modules that are written in Python. It is easy to simulate network behaviours using virtual network components, including routers, switches, and hosts, which are described as programmable classes. The network was coded to have three BGP routers that were linked by an internet exchange (IX), where each router was a representation of a local network that had web servers, user systems, and other services. The source code of the implementation and client interface of network management is presented in Figures 5 and 6, respectively.

Figure 5: Emulated Network Development.

AUTONOMOUS SYSTEMS AND INTERNET EXCHANGE POINT CONFIGURATION

As shown in Figure 6, the centre of the network configuration was the instantiation of Autonomous Systems (ASes), namely, asn_numbers, or, in this case, AS150, AS151, and AS152. These systems are isolated networks that have the ability to route independently. Each of the AS has one router, router0, router1, and router2, respectively, indicating the gateway on which the network traffic is controlled and shared. These routers were then linked together via a particular internet exchange point (IXP), which is referred to as ix100, as shown in Figure 6 below. The IXP is an important factor in determining whether the ASes are able to share the internet traffic.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

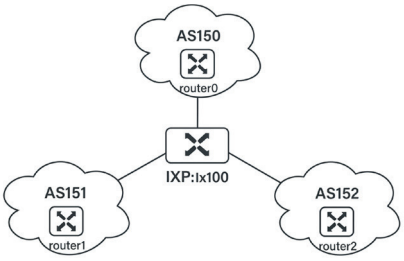

Figure 6: Autonomous Systems and Internet Exchange Point (IXP) configuration

BGP ROUTING IMPLEMENTATION

The IXP ASes were provided with the Border Gateway Protocol (BGP) peering sessions. This may be observed in Figure 7, with the following being the contents of the function: ebgp. RsPeer(ix number, asn). This facilitated the distribution of routing information throughout the network. This configuration directly reflects the working principle of internet routing and is directly related to the complex network architecture in which the exchange of information between ASes is to optimise path choice and traffic flow.

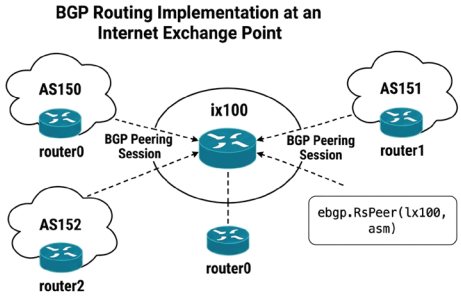

Figure 7: BGP Routing Implementation at an Internet Exchange Point

NETWORK MODULARITY

To simplify and modularise the process of constructing an Autonomous System (AS), a reusable configuration function was implemented. This function automates router configuration and establishes connectivity to the Internet Exchange Point (IXP), as illustrated in Figure 8.

```
#!/usr/bin/env python3
# encoding: utf-8
from seedemu.layers import Base, Routing, Ebgp
from seedemu.services import WebService
from seedemu.compiler import Docker
from seedemu.core import Emulator, Binding, Filter

# Function to create an autonomous system with a router connected to an IX
def setup_autonomous_system(base_layer, asn, router_name, ix_network):
    asys = base_layer.createAutonomousSystem(asn)
    asys.createRouter(router_name).joinNetwork(ix_network)
    return asys
```

Figure 8: Function for AS Configuration

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

NETWORK EXECUTION

Having established the various methods and the various network configurations, and the topology of the network, we used the compile method using the Docker argument, which is represented in Figure 9, to convert our abstract network topology to an executable emulation environment. The specified topology is transformed into a collection of interrelated Docker containers using this approach, with each of them executing one of the elements of a simulated network.

```
35    # Add layers to the emulator and render
36    emu.addlayer(base)
37    emu.addlayer(routing)
38    emu.addlayer(ebgp)
39    emu.render()
40
41    # Compile the emulation environment
42    emu.compile(Docker(), './output')
```

Figure 9: Emulated Network Compilation and Execution

Figure 10 presents the client interface of the network that is in compliance and is executed. Under this interface, we could get access to all the routers, visualise the connectivity, and the relationship of the network with the internet exchange point.

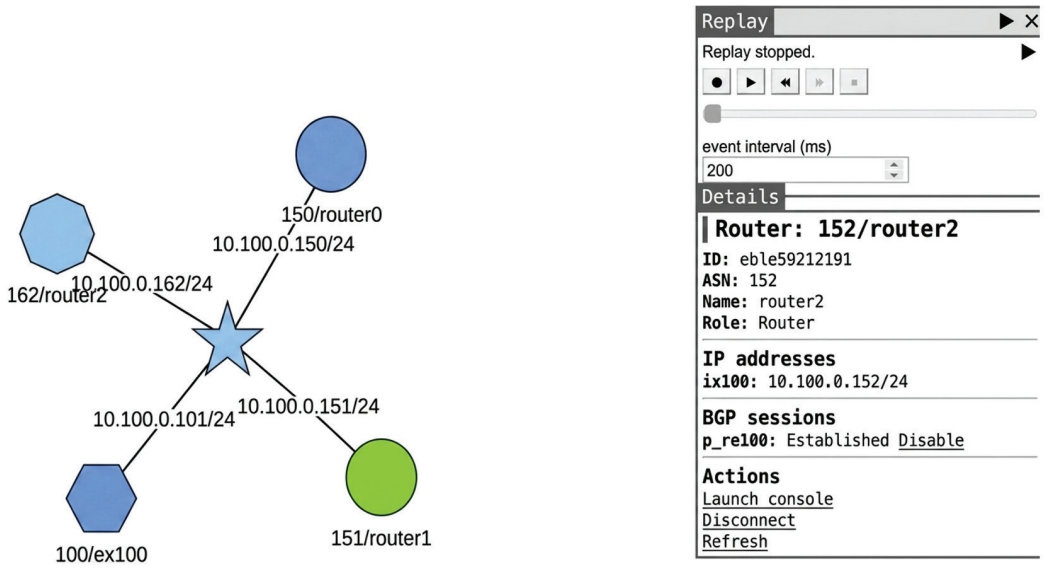

Figure 10: Client Interface for the Developed Emulated Network

DOS ATTACK SIMULATION

As our network environment was ready and operational, we created a sequence of DoS attacks that would be emulated to load stress on our network environment, and that would be utilized to test our proposed NIDPS model. The first one was a TCP SYN flood attack on a given IP address and port within the network. This attack is of a multithreaded nature, and it is an actual DDoS attack, which is a simulation, meaning that it floods the particular address with lots of packets, hoping to interfere with the normal functioning of the address. Figure 11 can be regarded as a logic of the packet configuration and thread sessions.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834


```
// Configure Security
SecurityHttpConfig(httpSecurityConfig, jwtAuthFilter, authenticationManager, userDetailsService,
        tokenProvider, passwordEncoder, securityProperties, tokenCookie, tokenHeader);

class SymbolController {
   def __init__(self, target_up, target_port):
       super(SymbolController, self).__init__()
       self.target_up = target_up
       self.target_port = target_port

   def run(self):
      try:
            # Craft & fire exploit
            payload = Payload(self.target_up, f"{PayloadImpl}.template", Flag="1")
            # Define how to keep reading results
            while True:
                resultCacheE =VerboseWriter(
                   Logging.Output("resultFile", wait_sec_packet, fself.target_upself.target_port}")
       except Exception as ex:
            # Logging error/wrong sending (no aspects) {E}
            print(f"error: {ex}")

 def main(target_up, target_port, thread_count):
    threads = []
    for _ in range(thread_count):
        thread = Symbol(target_up, target_port)
        thread.start()
        threads.append(thread)
    for thread in threads:
        thread.join()

if __name__ == "__main__":
    target_up = input('Enter target IP: ')
```

Figure 11: TCP Syn Flood Attack.

Besides this, we went ahead to devise a UDP flood attack as in Figure 12, with the intention of flooding the network bandwidth, consuming and exhausting server resources. Here, we determined the size of the packets being sent, the threads, and the IP address and port of a particular IP.

- Packet Sending: Sending packets to a destination is performed by the script with a large number of threads running simultaneously. All threads flood packets to the limit of their capacity to the network and the target server.
- Packet Size: Packet size is set to 1024 bytes.
- Target Specification: The Packet is transmitted to a targeted IP address and port, which makes the attack focused and may be more disruptive.

```
THREAD_COUNT = 1000  # number of threads to use in the attack

def send_udp_packets():
    """
    Function to send UDP packets to the target and log each packet sent.
    """
    # Create a socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # Generate random bytes to send
    bytes_to_send = random._urandom(PACKET_SIZE)

    while True:
        try:
            # Send the packet to the target
            sock.sendto(bytes_to_send, (TARGET_IP, TARGET_PORT))
            logging.info(f"Sent (PACKET_SIZE} bytes to {TARGET_IP):{TARGET_PORT}")
        except Exception as e:
            logging.error(f"Error: {e}")
            break

def main():
    """
    Main function to start the UDP flood attack and log the start and end of the attack.
    """
    logging.info(f"Starting UDP flood attack on {TARGET_IP):{TARGET_PORT} with {THREAD_COUNT}
    threads = []

    for _ in range(THREAD_COUNT):
        # Create a new thread to send UDP packets
        thread = threading.Thread(target=send_udp_packets)
        thread.start()
```

Figure 12: UDP Flood Attack

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

NIDPS MODEL DEVELOPMENT

In this part, we have presented detailed information about the various modules that have been created towards a complete intrusion detection and prevention system. The NIDPS will include the IDS module, where the threats are identified, which are recorded to be investigated by the network administrator. This was in turn succeeded by a long period of functionality, IPS, where the threats identified do not penetrate into the network. The last step of the model was to put in place an alert system that will alert the network administrator and a web interface to manage the system. The drawn sequence of activities illustrated in Figure 13 reflects simulated DoS attacks directed to the emulated network, the role of the NIDPS in the detection of the attack, and the timely alerting of the network administrator to take the required measures, which can be any of the following; blocking the IP addresses where the attacks originated, closing open ports and amending firewall rules.

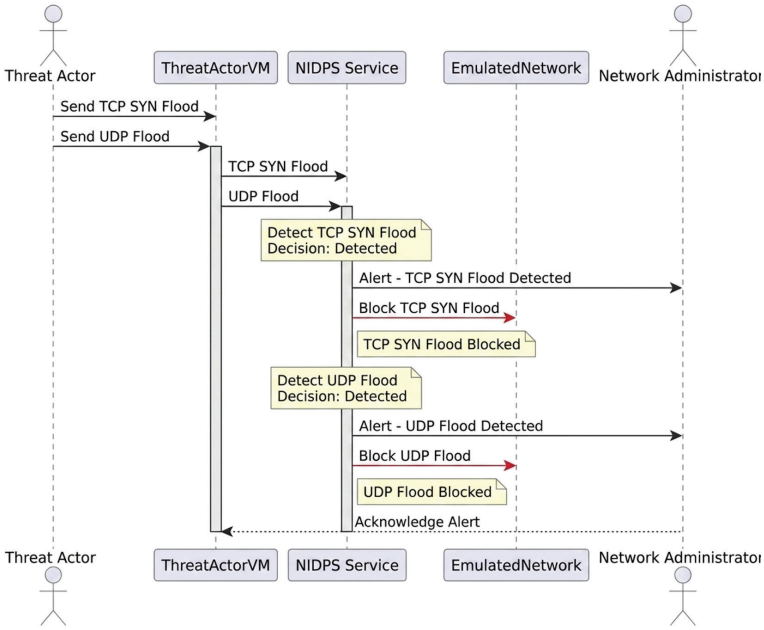

Figure 13: DoS Attack and NIDPS Sequence Diagram

NIDPS MODULE

To have the IDS module work as expected, we have gathered packet traffic data on the basis of simulated normal traffic and DoS attack traffic that was captured using the Wireshark tool and a custom-built Python-based packet sniffer script. The script also made use of the library known as PyShark, which captured and analysed network traffic. As has been mentioned above, we simulated two DoS security attacks, tcpsynflood.py and udpflood.py. Features were extracted by the packet sniffer and analysers, depending on both simulated attacks in the CICFlowmeter open source application. The resulting records were abridged into TCPSYNFloodDataset.csv and UD PfloodDataset.csv, respectively. The table and graphical distribution of the types of attack are as illustrated in Table 1 and Figure 14, respectively.

Table 1: Dataset Distribution of Traffic Types

Traffic Type	Count
Normal	7,000
TCP SYN Flood	5,496
UDP Flood	5,090

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

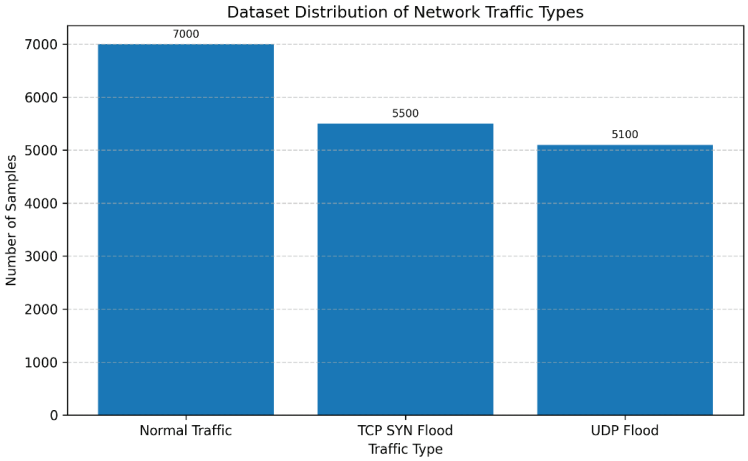

Figure 14: Dataset Distribution

The second stage of development implied cleaning the features that had been extracted and preprocessing them with pre-defined functions and libraries. A column with a label was added to each dataset that contained the normal traffic dataset, and empty fields were deleted. This was in order to embrace a supervised learning method of the ML model. We have continued with the label encoding process wherein we called the LabelEncoder() module to encode the categorical data into numerical ones. Since the packet sniffing and feature extraction steps were already done, the dataset generation and preprocessing steps were also done. The following step in the development of the NIDPS was to train and test the ensemble ML models using the dataset. One of the most important steps of the model development was the feature selection, as it is important to determine the important and relevant features to be used to train the model. Firstly, the Recursive Feature Elimination with Cross Validation (RFECV) method was embraced in order to determine significant features to be included in the training. This method of feature selection applies random forest as an estimator that recursively adopts the features in order to decide the relevance of the features using a set of pre-defined characteristics. In this step, cross-validation goes on to remove irrelevant features and minimise data noise. As far as data classification in IDS is concerned, RFECV boasts of improved performance (Nguyen, 2022). The RFECV was capable of identifying and reducing the number of features to be used in training, among 82 features, to 10 features. Figure 15 is the importance scale of each feature. Moreover, an experimental comparison was adopted with another feature selection process. The Correlation-based Feature Selection method in this case will pick the subset of features that will give more information. The method that has been found to enhance the performance of classifiers such as DT and SVM (Farahani, 2020) is based on statistical correlation among features that eliminates redundant features. Through this, 47 features were picked among 82 features that had the least interdependence with each other. The correlation of features is given in Figure 16.

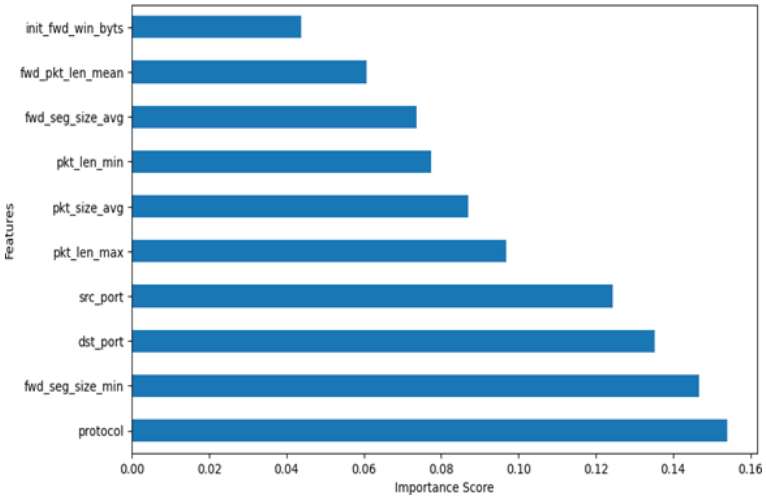

Figure 15. Feature Importances of Selected Features by RFECV

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

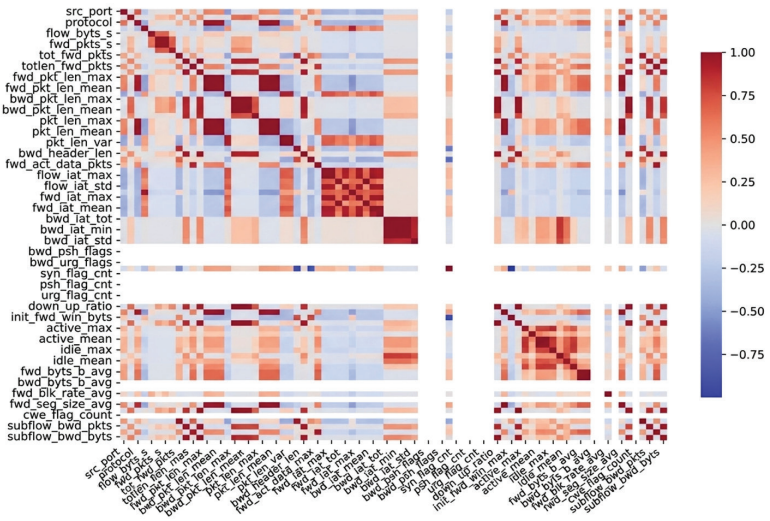

Figure 16. Correlation Matrix of Features

After applying feature selection techniques and identifying features to use with both methods, DT, XGBoost, and SVM classifiers were trained and tested individually, and finally, the voting ensemble consisting of the three classifiers was used to combine their predictive value in identifying malicious and benign traffic. Figure 17 shows the ordered diagram of the NIDPS model, beginning with the packet sniffing process until the deployment of the model. Another facility was implemented on the model, and that is the email alert generation and the IPS module. Using these, when the NIDPS identifies malicious traffic across the network, it will create an email message that will be sent to the network administrator, and the network administrator is then allowed to invoke the IPS feature. This gave the administrator authority to view the specifics of the network traffic and determine the course of action they can take, either to block or permit.

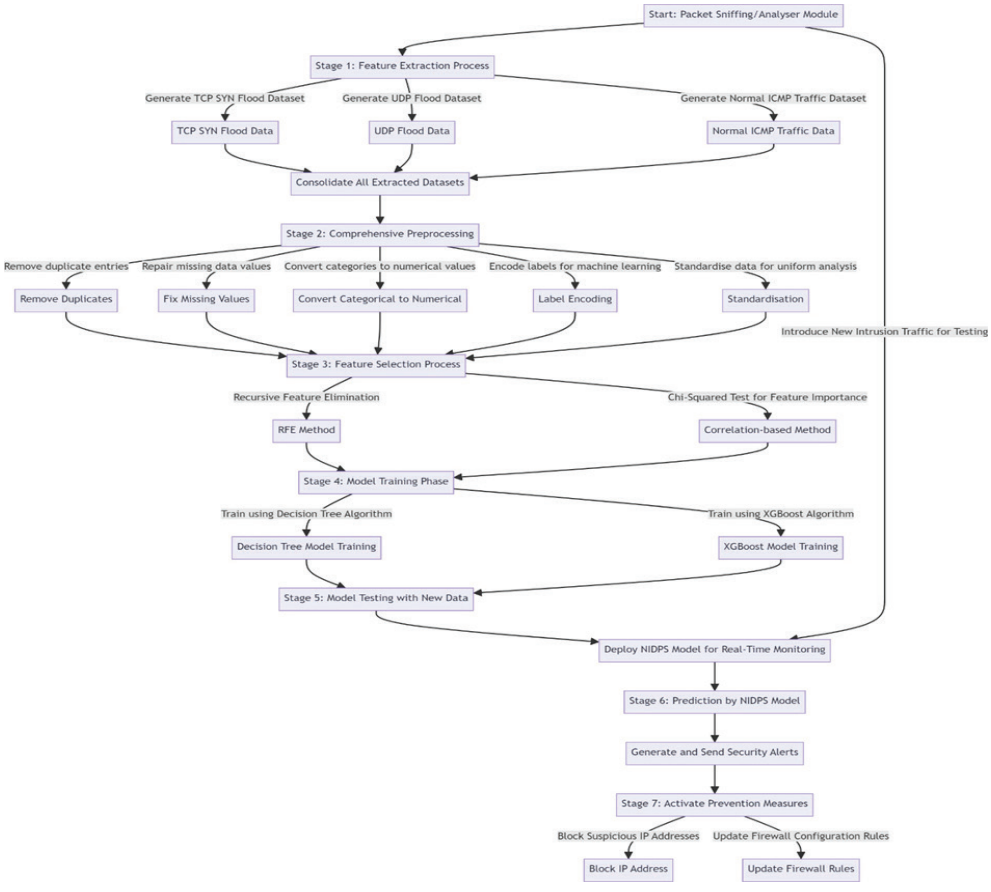

Figure 17: NIDPS Model Training and Operation Architecture

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

SECURITY ALERT GENERATION

This stage of the model is when the model has forecasted an intrusion following its strict training and testing. The alert generation functionality sends an email notification to the network administrator of an intrusion and a brief perspective of what kind of attack it is, the time of attack, the source and target IP addresses, and the destination port number. This provides the administrator with a short understanding of the intrusion and can take the required measures where necessary. Fig. 18 is the email reply that the administrator will receive when the model predicts that there is an intrusion in the network.

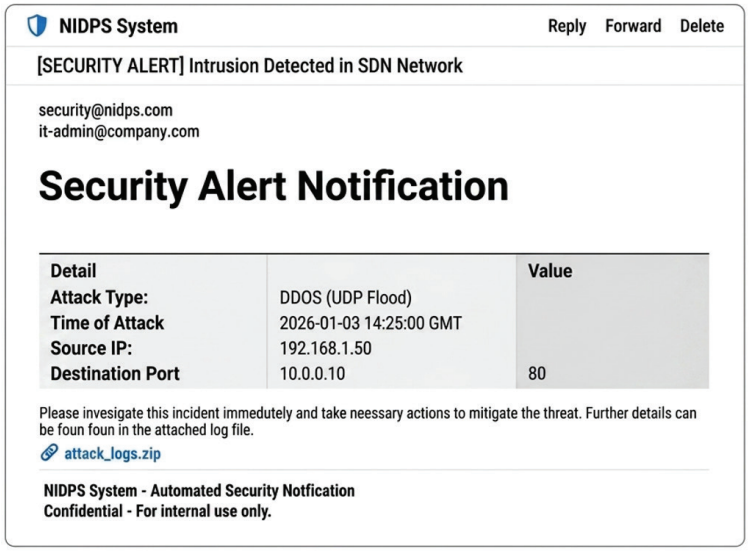

Figure 18: Security Alert Notification

3.0 RESULTS

In this part of the research paper, a close analysis of the findings and an assessment of the suggested NIDPS model will be given. Our model has been developed on the basis of three big experiments. Our model was an ensemble of three ML classifiers (DT, XGBoost, and SVM). To assess the NIDPS model, a number of metrics were taken into consideration, particularly in light of the importance of reducing false positives and the advanced speed of detecting real intrusions within the network.

A. Precision: This measures the accuracy of the positive predictions made by the model; it essentially quantifies the number of true positive results in the context of all positive predictions made (including both true positives and false positives). This metric can be represented in a mathematical formula as seen below:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \times 100\ \%$$

B. Accuracy: This represents the overall correctness of the model. This is calculated as the ratio of correctly predicted observations (both true positives and true negatives) to all observations in the dataset. It can be mathematically expressed as seen below:

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negative}}{\text{Total number of observations(samples)}}$$

C. Recall: Recall, or sensitivity, measures the model's ability to correctly identify all actual positives from the data, thereby assessing the model's capability to find all relevant instances. This is particularly critical to identifying actual threats, reducing the possibility of missing threats to a negligible level. The mathematical representation can be seen below.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positive + False Negative}} \times 100\ \%$$

D. F1-Score (F1-S): This provides a balance between Precision and Recall, which is particularly beneficial in situations where there is an imbalanced data distribution. It is the harmonic mean of Precision and Recall.

$$\text{Recall} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision + Recall}} \times 100\ \%$$

In addition to these metrics, system throughput and resource usage are also measured to determine the impact of the implementation of the NIDPS on the network, as the aim is to ensure network performance is not significantly traded off for network security.

ENSEMBLE-BASED NIDPS WITHOUT FEATURE SELECTION

According to section 4, 82 features were picked out of the data file (.pcap file) of network traffic information received using the packet sniffer module of the NIDPS. These 82 features were directly used in training the individual base classifiers of the ensemble model, with all of them having considerably high training and testing accuracy as observed in Table 2. We went on to apply the voting ensemble that uses the majority voting method, whereby it capitalizes on the strong aspects of the base classifiers and offers a tradeoff of their weak aspects to improve model prediction. Under this ensemble method, we took a mild voting strategy. This is achieved by using the maximum prediction probability of the base classifiers. Table 2 shows that the voting ensemble obtained a perfect score on all metrics with no feature selection method implemented, whereas Figure 19 provides a confusion matrix associated with this approach. Furthermore, the time taken by the ensemble model to make predictions was 0.065 seconds, as observed in Table 5.

Table 2: Performance Evaluation of the Ensemble in Comparison with Each Classifier

Model	Accuracy	Precision	Recall	F1 Score
Decision Tree	1.0000	1.0000	1.0000	1.0000
XGBoost	1.0000	1.0000	1.0000	1.0000
SVM	0.9994	0.9994	0.9994	0.9994
Ensemble	1.0000	1.0000	1.0000	1.0000

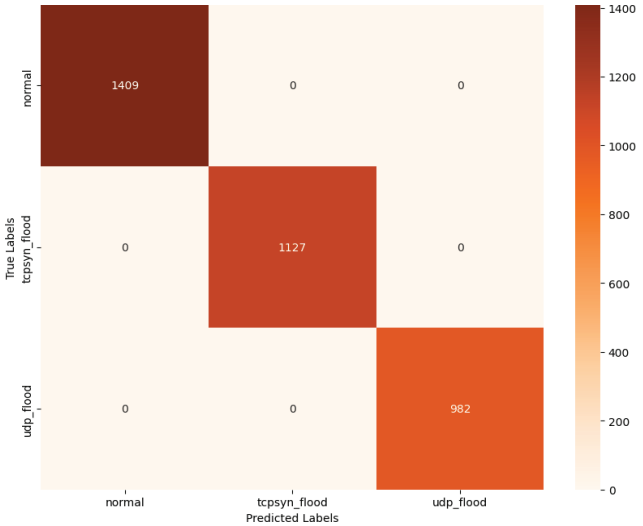

Figure 19: Confusion Matrix for the Ensemble Model without Feature Selection Method

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

ENSEMBLE-BASED NIDPS WITH RFE

The ensemble-based NIDPS with RFE made use of 10 features picked by the dataset using the recursive feature elimination correlation validation feature selection procedure. In this method of selecting features, the random forest classifier is used. The method was also producing high results in the various performance measures, such as precision, accuracy, recall, and F1-score, that are displayed in Table 3. Moreover, the value of each feature that is significant in the model prediction can be illustrated visually in Figure 20. The confusion matrix in Figure 17 also gives more insights into this technique. The ensemble model that used RFE also predicted the test data in 0.0358 seconds, as shown in Table 5.

Table 3: Performance Evaluation of the Ensemble Model and Standalone Classifiers with RFE

Model	Accuracy	Precision	Recall	F1 Score
Decision Tree	1.0000	1.0000	1.0000	1.0000
XGBoost	1.0000	1.0000	1.0000	1.0000
SVM	1.0000	1.0000	1.0000	1.0000
Ensemble	1.0000	1.0000	1.0000	1.0000

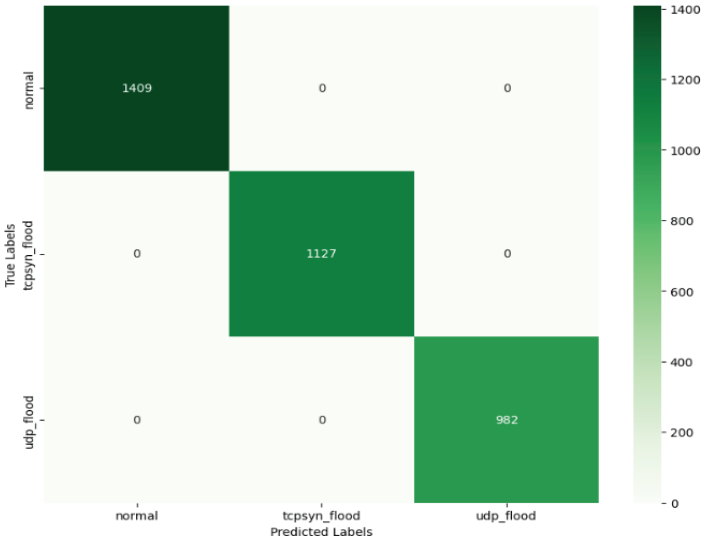

Figure 20: Confusion Matrix for the Ensemble Model with the Recursive Feature Elimination Method

ENSEMBLE-BASED NIDPS WITH CORRELATION-BASED FEATURE SELECTION

The feature selection method was also the correlation-based method, which was used to further test the performance of the ensemble model against the other two model implementation methods. The feature interdependency threshold was 0.9, which was a high threshold. This threshold allowed us to filter out features that are more interdependent by only selecting those features that are below the threshold, and we were left with 47 statistically unique features. The second step was to continue with the ensemble model training with the chosen features, and once again, the performance of the model was excellent, with all the classifiers having a 100 percent accuracy except the SVM with 99.94 percent, as observed in Table 4. The confusion matrix in Figure 21 provides more information on the matter, whereas Table 4 illustrates the various measures of performance. Also, the speed of prediction on this model was taken into consideration, as observed in the ensemble comparison table in Table 5.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

Table 4: Performance Evaluation of the Ensemble Model and Standalone Classifiers with Correlation-Based Feature Selection

Model	Accuracy	Precision	Recall	F1 Score
Decision Tree	1.0000	1.0000	1.0000	1.0000
XGBoost	1.0000	1.0000	1.0000	1.0000
SVM	0.9994	0.9994	0.9994	0.9994
Ensemble	1.0000	1.0000	1.0000	1.0000

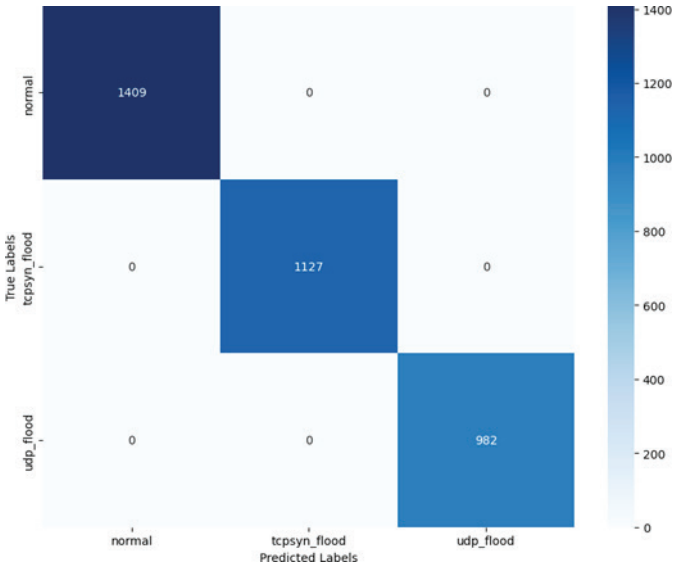

Figure 21: Confusion Matrix for the Ensemble Model with Correlation-Based Feature Selection Method

PERFORMANCE COMPARISON OF THE ENSEMBLE MODELS

Since the various ensemble models have been applied with and without the use of various feature selection methods, it is necessary and urgent to test the effectiveness of each of the models applied. We have not only compared the general performance measures of each model in our analysis, but have also noted the computational efficiency of the models, particularly the speed of prediction. This is an evaluation that is presented in Table 5, which gives an in-depth perspective of the NIDPS ensemble model performances. Besides that, figures 22 and 23 demonstrate the graphic depiction of such performance comparisons. The comparisons provide an insight into the comprehension of what models provide the best tradeoff between detection performances and the ability to operate, particularly in the case of implementation in a real-time network environment.

Table 5: Performance Comparison of the Ensemble Models in relation to Computational Speed

Model	Accuracy	Precision	Recall	F1 Score	Prediction Speed
Ensemble Model without Feature Selection	1.0000	1.0000	1.0000	1.0000	0.0755
Ensemble Model with RFE	1.0000	1.0000	1.0000	1.0000	0.0358
Ensemble Model with Correlation-Based Feature Selection	1.0000	1.0000	1.0000	1.0000	0.127

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

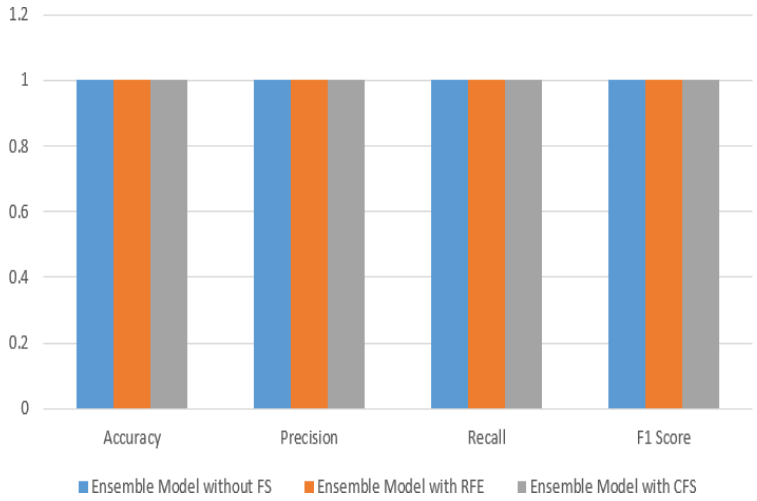

Figure 22: Performance Metrics Comparison for the Ensemble Models

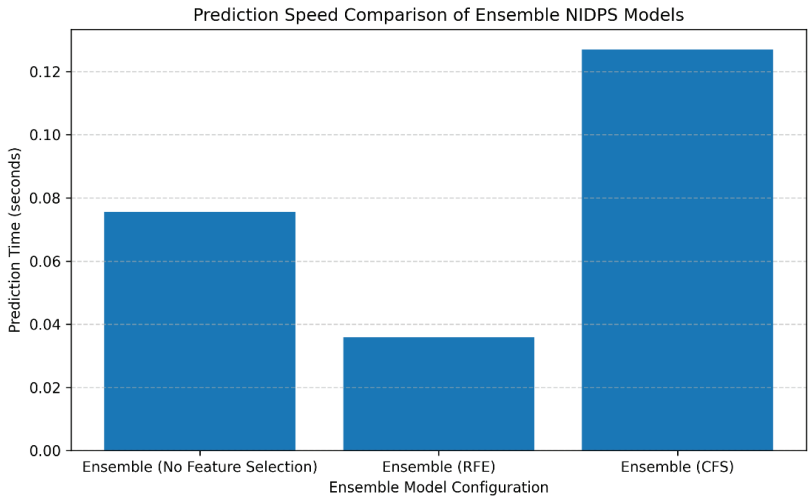

Figure 23: Prediction Speed for each Ensemble Model Implementation

DISCUSSION

The ensemble-based NIDPS system proposed in this paper demonstrated a high level of detection during all of the experiments performed in three experimental environments: no feature selection, Recursive Feature Elimination (RFE), and Correlation-Based Feature Selection. While Denial-of-Service (DoS) attacks were the main focus of this study, intrusion detection systems in real-world situations encounter a far broader range of attacks, including probing, privilege escalation, malware propagation, and zero-day threats. Due to its reproducibility, impact on traffic, and suitability for controlled experimentation in an SDN context, denial-of-service (DoS) attacks were our primary emphasis instead of attempting to capture every possible sort of attack. Having an almost zero difference in the recall and precision of a classifier, all ensemble variants achieved a 100 percent accuracy, which shows the versatility and stability of ensemble learning in a cybersecurity environment.

A. Ensemble Model with no Feature Selection

The ensemble model without feature selection showed the best classification performance, with all accuracy, precision, recall, and F1 score equal to 1.000, which shows that the combination of the classifiers (Decision Tree, XGBoost, and SVM) was able to learn using the full set of 82 features. However, the crude model had a relatively high prediction latency (0.0755 seconds), which can limit its use in real-time network applications, especially when the network is experiencing heavy traffic.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

B. Recursive Feature Elimination (RFE)

The usage of RFE, a wrapper-based feature selection approach, reduced the number of features by 10 important ones and maintained the same classification accuracy. The RFE-based model had superior computational performance, and its fastest rate of prediction was 0.0358 seconds. This shows that RFE can be useful in reducing dimensions and improving performance, which is in line with earlier studies that point to the ability of the method to enhance the generalisability of a model and efficiency in high-dimensional data scenarios.

C. Correlation-Based Feature Selection

The third experimental setting, which featured Correlation-Based Feature Selection, also gave excellent results. This arrangement was a little longer than predicted (0.127 seconds). This method gave a satisfactory tradeoff between precision and a middle range shrinkage of features, although it retained 47 features (an artifact of its focus on reducing redundancy instead of aggressively dimensionality reduction). The reduced feature interdependence supported model interpretability and resilience, especially when dealing with the multicollinear effects of network traffic data.

KEY FINDINGS

These findings, combined with each other, bear witness to some significant observations:

A. Reliability of Models Using Different Feature Approaches

The integration of several classifiers enhances the limits of decisions and offsets the shortcomings of the single models. The ensemble model was more robust in the detection and classification of DoS attacks, independent of the method of feature selection.

B. Feature Selection Tradeoffs

Even though each of the methods was very precise, RFE offered the best compromise of speed and computing economy. This tradeoff is particularly crucial in environments where the resources are few, such as SDNs founded on the Internet of Things or on edge networks.

C. Real-Time Readiness

The latency of all variations of NIDPS is very low, and it can be used in real-time. This system will be best suited in a live network environment as it can be used to support administrative decisions by providing integrated IPS capabilities, as well as raising alerts at the appropriate time.

D. Scalability and Emulation Effectiveness

The validity of the scalability and operational reliability of the model in the face of controlled DoS attacks was tested by using the SEED internet emulator to recreate a distributed network environment in a realistic way. This is further supported by the adoption of enterprise programmable networks at scale.

LIMITATIONS AND FUTURE DIRECTIONS

The encouraging results have a number of caveats that must be taken into consideration. The experiments were first conducted on a simulated controlled dataset; to further generalise, it would be required to conduct the experiments on other forms of traffic, e.g., benign outliers and more complex multi-stage attacks. Secondly, soft vote aggregation enhanced the consensus of classifiers, but fine-tuning prediction could be enhanced with further research on dynamic ensemble optimisation systems such as meta-learning or adaptive boosting in the face of an evolving threat. In order to make the recommended NIDPS more representative of real-world network situations, further research will include evaluating it against more types of attacks and using datasets with significant imbalances.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

CONCLUSION

We have described the process of designing and evaluating a Network Intrusion Detection and Prevention System (NIDPS) in this study that employs an ensemble strategy and would run within an SDN environment. It was demonstrated that the proposed system was highly sensitive to detecting Denial-of-Service (DoS) attacks, computationally efficient, and responsive in real-time because it employed a hybrid method that integrates different machine learning classifiers - Decision Tree, XGBoost, and Support Vector Machine - with sophisticated feature selection methods. The goal of all three experiments was to reach almost perfect or ideal detection values, and the result is what it accomplished using full features, Recursive Feature Elimination (RFE), and Correlation-Based Feature Selection. In the context of real-time network monitoring schemes, however, the ensemble model based on RFE really came into its own owing to its blistering prompt responses. We find that we can achieve increased scalability, responsiveness, and detection in the system by using a convergence of feature optimisation and ensemble learning. Simulation of the architecture on real-life network infrastructures and attack scenarios with the SEED Internet Emulator demonstrated the practicality of the suggested architecture. Moreover, the real-time production of alert features, as well as preventive functionality, is incorporated into the problem response process that gives network managers a response to an action. Its strengths notwithstanding, the current model has primarily been applied to controlled DoS traffic. The future research objectives should be to evaluate the system against a broader range of attacks, such as zero-day threats, in addition to putting the research through its paces in real-world SDN testbeds to understand how it will respond to changes over time. Other potential improvements are the integration of deep learning, federated learning in the case of distributed context, and dynamic reconfiguring of ensembles.

REFERENCES

[1] H. LI, Q. LI, Z. XU, and X. YE, "Digital Technologies," *Journal of Digital Economy*, vol. 3, pp. 240–248, Mar. 2025, doi: https://doi.org/10.1016/j.jdec.2025.02.001.

[2] O. B. Adoghe, B. A. Ikharo, H. E. Amhenrior, J. Abiodun, and E. B. Chukwu, "Data Collection Module for a Centralized Electronic Health Records System," *Journal of Engineering Research Innovation and Scientific Development (JERISD)*, vol. 2, no. 1, pp. 11–19, Mar. 2024, doi: https://doi.org/10.61448/jerisd21242.

[3] R. Kaur, D. Gabrijelčič, and T. Klobučar, "Artificial intelligence for cybersecurity: Literature review and future research directions," *Information Fusion*, vol. 97, no. 101804, pp. 1–29, 2023, doi: https://doi.org/10.1016/j.inffus.2023.101804.

[4] R. Muggah and M. Margolis, "Why we need global rules to crack down on cybercrime," *World Economic Forum*, Jan. 02, 2023. https://www.weforum.org/stories/2023/01/global-rules-crack-down-cybercrime/ (accessed Apr. 22, 2025).

[5] S. Morgan, "Cybercrime to cost the world $10.5 trillion annually by 2025," *Cybercrime Magazine*, Nov. 13, 2020. https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/ (accessed Apr. 22, 2025).

[6] L. Diana, P. Dini, and D. Paolini, "Overview on Intrusion Detection Systems for Computers Networking Security," *Computers*, vol. 14, no. 3, p. 87, Mar. 2025, doi: https://doi.org/10.3390/computers14030087.

[7] M. Rahman, S. A. Shakil, and M. R. Mustakim, "A Survey on Intrusion Detection System in IoT Networks," *Cyber Security and Applications*, p. 100082, Dec. 2024, doi: https://doi.org/10.1016/j.csa.2024.100082.

[8] O. Joseph and R. Ajax, "Comparison of Traditional vs. AI-Based Intrusion Detection and Prevention Systems: Efficiency and Accuracy," *ResearchGate*, Mar. 10,

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

2025. https://www.researchgate.net/publication/389717300_Comparison_of_Traditional_vs_AI-Based_Intrusion_Detection_and_Prevention_Systems_Efficiency_and_Accuracy (accessed Apr. 22, 2025).

[9] H. Dhrir, M. Charfeddine, N. Tarhouni, and H. M. Kammoun, "Machine learning- and deep learning-based anomaly detection in firewalls: a survey," *The Journal of Supercomputing*, vol. 81, no. 6, Apr. 2025, doi: https://doi.org/10.1007/s11227-025-07212-y.

[10] S. Muneer, U. Farooq, A. Athar, M. A. Raza, T. M. Ghazal, and S. Sakib, "A Critical Review of Artificial Intelligence Based Approaches in Intrusion Detection: A Comprehensive Analysis," *Journal of engineering*, vol. 2024, pp. 1–16, Apr. 2024, doi: https://doi.org/10.1155/2024/3909173.

[11] T. Tong and Z. Li, "Predicting learning achievement using ensemble learning with result explanation," *PLoS ONE*, vol. 20, no. 1, pp. e0312124–e0312124, Jan. 2025, doi: https://doi.org/10.1371/journal.pone.0312124.

[12] Kristel, A.-F. Rutkowski, and C. Saunders, "The whole of cyber defense: Syncing practice and theory," *The Journal of Strategic Information Systems*, vol. 33, no. 4, pp. 101861–101861, Sep. 2024, doi: https://doi.org/10.1016/j.jsis.2024.101861.

[13] N. James, "How Many Cyber Attacks Per Day: The Latest Stats and Impacts in 2023 - Astra Security Blog," *ASTRA*, Apr. 18, 2023. https://www.getastra.com/blog/security-audit/how-many-cyber-attacks-per-day/ (accessed Apr. 22, 2025).

[14] C. Rajathi and P. Rukmani, "Hybrid Learning Model for intrusion detection system: A combination of parametric and non-parametric classifiers," *Alexandria Engineering Journal*, vol. 112, pp. 384–396, Nov. 2024, doi: https://doi.org/10.1016/j.aej.2024.10.101.

[15] K. Noor, Agbotiname Lucky Imoize, C.-T. Li, and C.-Y. Weng, "A Review of Machine Learning and Transfer Learning Strategies for Intrusion Detection Systems in 5G and Beyond," *Mathematics*, vol. 13, no. 7, pp. 1088–1088, Mar. 2025, doi: https://doi.org/10.3390/math13071088.

[16] T. Sowmya and E.A. Mary Anita, "A comprehensive review of AI based intrusion detection system," *ScienceDirect*, vol. 28, pp. 100827–100827, Jun. 2023, doi: https://doi.org/10.1016/j.measen.2023.100827.

[17] P. A. doost, S. S. Moghadam, E. Khezri, A. Basem, and M. Trik, "A new intrusion detection method using ensemble classification and feature selection," *Scientific Reports*, vol. 15, no. 1, Apr. 2025, doi: https://doi.org/10.1038/s41598-025-98604-w.

[18] U. Ahmed *et al.*, "Signature-based intrusion detection using machine learning and deep learning approaches empowered with fuzzy clustering," *Scientific Reports*, vol. 15, no. 1, Jan. 2025, doi: https://doi.org/10.1038/s41598-025-85866-7.

[19] S. L. Pawar and M. S. Karyakarte, "A Hybrid CNN and Attentive Hierarchical BiLSTM Model with SMO for Intrusion Detection in IIoT," Apr. 2025, doi: https://doi.org/10.21203/rs.3.rs-6158243/v1.

[20] J. Xie, "Application Study on the Reinforcement Learning Strategies in the Network Awareness Risk Perception and Prevention,"*The International journal of computational intelligence systems/International journal of computational intelligence systems*, vol. 17, no. 1, May 2024, doi: https://doi.org/10.1007/s44196-024-00492-x.

[21] B. Susilo, A. Muis, and R. F. Sari, "Intelligent Intrusion Detection System Against Various Attacks Based on a Hybrid Deep Learning Algorithm," *Sensors*, vol. 25, no.

http://dx.doi.org/10.21622/ACE.2026.06.1.1834

2, pp. 580–580, Jan. 2025, doi: https://doi.org/10.3390/s25020580.

[22] S. Walling and S. Lodh, "Enhancing IoT intrusion detection through machine learning with AN-SFS: a novel approach to high performing adaptive feature selection," *Discover Internet of Things*, vol. 4, no. 1, Oct. 2024, doi: https://doi.org/10.1007/s43926-024-00074-5.

[23] N. Pudjihartono, T. Fadason, A. W. Kempa-Liehr, and J. M. O'Sullivan, "A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction," *Frontiers in Bioinformatics*, vol. 2, Jun. 2022, doi: https://doi.org/10.3389/fbinf.2022.927312.

[24] N. Anand, R. Sehgal, S. Anand, and A. Kaushik, "Feature selection on educational data using Boruta algorithm," *International Journal of Computational Intelligence Studies*, vol. 10, no. 1, p. 27, 2021, doi: https://doi.org/10.1504/ijcistudies.2021.113826.

[25] A. A. Khan, O. Chaudhari, and R. Chandra, "A Review of Ensemble Learning and Data Augmentation Models for Class Imbalanced problems: Combination, Implementation and Evaluation," *Expert Systems with Applications*, vol. 244, no. 122778, p. 122778, Dec. 2023, doi: https://doi.org/10.1016/j.eswa.2023.122778.

[26] A. M. Alsaffar, M. Nouri-Baygi, and H. M. Zolbanin, "Shielding networks: enhancing intrusion detection with hybrid feature selection and stack ensemble learning," *Journal of Big Data*, vol. 11, no. 1, Sep. 2024, doi: https://doi.org/10.1186/s40537-024-00994-7.

[27] H. Duy, "Recursive Feature Elimination Algorithm for Intrusion Detection Systems," *2022 7th International Conference on Business and Industrial Research (ICBIR)*, May 2022, doi: https://doi.org/10.1109/icbir54589.2022.9786411.

[28] A. Fausto, G. Gaggero, F. Patrone, and M. Marchese, "Reduction of the Delays Within an Intrusion Detection System (IDS) Based on Software Defined Networking (SDN)," *IEEE Access*, vol. 10, pp. 109850–109862, 2022, doi: https://doi.org/10.1109/access.2022.3214974.