IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 November 2023, accepted 4 December 2023, date of publication 28 December 2023,
date of current version 9 January 2024.

*Digital Object Identifier 10.1109/ACCESS.2023.3347778*

RESEARCH ARTICLE

# Artificial Intelligence-Based Intrusion Detection and Prevention in Edge-Assisted SDWSN With Modified Honeycomb Structure

**JOSEPH KIPONGO[1], THEO G. SWART[1], (Senior Member, IEEE), AND EBENEZER ESENOGHO[1,2], (Member, IEEE)**

[1]Center for Telecommunications, Department of Electrical and Electronic Engineering Science, University of Johannesburg, Johannesburg 2006, South Africa
[2]Department of Electrical and Electronic Engineering, University of Botswana, Gaborone, Botswana

Corresponding authors: Ebenezer Esenogho (drebenic4real@gmail.com) and Theo G. Swart (tgswart@uj.ac.za)

**ABSTRACT** The software-defined wireless sensor network (SDWSN) has the potential to improve flexibility, scalability, and network performance, but security and quality of service (QoS) are major challenges due to attackers, poor network management, and inefficient route selection. Several existing works for intrusion detection had drawbacks like poor security, inefficient network management, higher energy consumption and latency, and lesser throughput. A modified honeycomb structure-based intrusion detection system for SDWSN is proposed to address these challenges, which includes secure authentication using the 3D cube algorithm, modified honeycomb-based network partitioning, clustering, reinforcement learning-based intelligent routing with a transfer learning-based deep Q network (TLDQN), and a hybrid intrusion detection system. The latter detects malicious nodes using a driver training-based optimization (DTO) algorithm and intrusions with a bidirectional generative adversarial network (Bi-GAN). The results show that the proposed system outperforms existing solutions in terms of security, network performance, and efficiency. The simulation of this research is conducted by NS-3.26 Network Simulator, and the performances are evaluated based on various performance metrics (with respect to the total number of nodes) like energy consumption, latency, throughput, packet delivery ratio, network lifetime, computation overhead, detection accuracy, packet drop ratio, and control overhead, which proved that the proposed work achieves superior performance compared to existing works. The evaluation also includes a total simulation period during which the system's real-time performance was conducted. Time-based metrics such as precision, recall, and F1-score, as well as confusion matrices, are utilized to analyze the system's effectiveness in real-time in response to dynamic network threats.

**INDEX TERMS** SDWSN, 3D cube, intrusion detection, secure routing, modified honeycomb-based network construction, authentication.

## I. INTRODUCTION

Wireless sensor network (WSN) is a group of sensors placed at various locations that monitor and record the environment and physical conditions of the network through the internet. The development of the Internet of Things (IoT) has increased the possibilities for WSN demand, further refining the continuing research in this field. WSN plays a vital role in the development of the IoT [1], which contains low-cost and low-computing resources. As a result, it is limited by low resource utilization and high energy consumption issues [2], [3]. To overcome these issues, a software-defined network is integrated with the WSN environment, which is known as a software defined wireless sensor network (SDWSN). The SDN provides scalable and flexible network management that divides the network into four planes: the data plane, the switch plane, the control plane, and the application plane. Generally, the SDWSN faces many issues in terms

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

© 2023 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

of security, energy consumption, and latency. Hence, routing is an essential process to reduce energy consumption and latency [4], [5], [6], [7].

The optimal path is selected between the sender (sensor node) and receiver (base station/sink). The existing works used many optimization algorithms such as particle swarm optimization (PSO), genetic algorithm (GA), artificial bee colony optimization (ABCO) and butterfly optimization for detecting the best routing path, which leads to a high packet loss rate and low throughput due to inefficient route detection [8], [9], [10]. To overcome this issue, reinforcement learning (RL)-based methods are proposed for selecting optimal routes that learn the environment automatically and select the best route for data transmission. Security is another important process in routing that helps increase the delivery ratio. Hence, some systems perform secure routing by considering the trust values for the next forwarder, which provides secure data transmission, but the secure routing process only preserves the network from outside attacks and does not address the inside attackers that perform malicious processes [11], [12], [13].

The intrusion detection system (IDS) is used to identify abnormal behavior of sensor nodes in order to resolve security issues in SDWSN [14]. The IDS includes two types of intrusions, namely signature-based IDS (SIDS) and anomaly-based IDS (AIDS). The SIDS detects known attacks by matching the flow features of the data packets, while AIDS detects unknown attacks by validating the packet features of the data packets. Existing works employ machine learning (ML) algorithms, such as those used to detect intrusions; however, they focus solely on malicious node detection and do not validate message integrity, increasing security breaches [15], [16], [17], [18], [19], [20]. Additionally, the information stored in the centralized SDN controller leads to a single point of failure and high complexity. For that purpose, multiple controllers are placed in the environment with the aid of blockchain; hence, all the transactions are stored and retrieved with the help of blockchain, increasing the security level of the network [21], [22], [23]. The traditional blockchain has a linear structure, which increases energy consumption and limits scalability [24]. Because of the considerable consequences that they have for the dependability, functioning, and safety of these networks, security and quality of service (QoS) are of the utmost importance in the context of SDWSN. When it comes to SDWSNs, data protection, network functioning, resource efficiency, and compliance with regulatory standards are the top priorities for security and QoS. If these components are ignored, it may result in severe repercussions such as data breaches, service outages, and degraded system reliability, which may have wider-reaching implications for the dependability and efficacy of the SDWSN setup. To overcome this issue, this research used hierarchical multiple blockchains which increase scalability and lower energy consumption due to their hierarchical structure.

### A. MOTIVATIONS AND OBJECTIVES

The SDWSN environment faces many challenges, such as poor security, low throughput, and high energy consumption, which are addressed separately by the existing approaches; however, the optimal solutions to these challenges are not yet provided. The following challenges are addressed by this research:

- **Poor Security**: Most of the existing works proposed secure routing mechanisms for providing security that do not address the intrusions in the network; in addition, the trust value was considered for legitimacy and route selection, which also increases security threats. The secure table information and trust values were stored on the centralized server without providing security, which led to poor security.
- **Inefficient Network Management**: The existing work placed the sensor nodes in a random manner, which leads to high complexity in network management due to its unstable nature. Some existing works used centralized controllers for network management, which results in inefficient network management because it cannot handle a large amount of data processing in a real-time environment.
- **High Energy Consumption and Latency**: The sensor node sends its data to the sink nodes or base station directly, which increases energy consumption and latency due to the long distances. An inefficient routing process also increases the latency and energy consumption due to the increasing number of hop counts.
- **Low Throughput**: The existing work considers only limited parameters (energy, trust, and distance) for the routing process, which leads to inefficient routing that reduces the throughput ratio. In addition, insecure and static routing also reduce the rate of throughput.

We are motivated by the aforementioned issues in order to identify network intrusions with less latency and energy consumption in the SDWSN environment [25], [26]. This research addresses poor security, high energy consumption, latency, and low throughput in the SDWSN environment.

The objectives of this research are listed as follows:

- To ensure the authenticity of the sensor nodes by performing authentication, which protects against external attacks.
- To perform efficient network management and reduce energy consumption by performing network partitioning and clustering.
- To increase throughput and reduce packet loss rate, secure and intelligent routing is performed, which protects against routing attacks.
- To improve security and increase delivery ratios through multi-controller-based intrusion detection and prevention.
- To increase security and scalability, hierarchical-based multiple blockchains are utilized in the SDWSN environment.

### B. RESEARCH CONTRIBUTIONS

This research mainly focuses on providing security by identifying intrusions using blockchain and artificial intelligence techniques in the SDWSN environment. The major contributions of this research are listed as follows:

- A modified honeycomb structure-based IDS model or architecture in SDWSN is developed and proposed, as seen in Fig. 1, which includes four algorithms in the data plane: the 3D cube, TLQDN, DTO, and Bi-GAN.
- The 3D cube algorithm, which ensures secure authentication by confirming the validity of sensor nodes and users' privately held information through the utilization of blockchain technology, is developed.
- In order to improve network performance in real-time environments, a modified honeycomb-based network construction and clustering approach is implemented.
- The TLDQN algorithm is developed for secure and optimal routing identification. It utilizes transfer learning to enhance the speed and efficiency of the DQN algorithm.
- The behavior of sensor nodes is analyzed using the DTO algorithm to identify any malicious nodes, thereby strengthening security measures.
- The Bi-GAN algorithm is implemented and effectively detects both signature-based and anomaly-based intrusions, resulting in a high level of security and minimal packet loss.
- A comparison between our model and other similar models is conducted to assess the progress achieved. This assessment was based on the chosen parameters and their impact on the results.
- By analyzing and evaluating these parameters, the study was able to determine the extent of improvement made by our model in relation to the others. The improvement is that the proposed work, HieMulti-Block, has higher performance compared to the existing works in terms of energy consumption, latency, throughput, packet delivery ratio, network lifetime, computation overhead, and detection accuracy. The flowchart of the proposed system model is given in Fig. 2.

### C. PAPER ORGANIZATION

The remainder of this research is organized as follows: Section II illustrates the existing works and their research gaps or challenges which were solved by the proposed work. Section III explains the detailed description of the proposed methodology, algorithm, pseudocode, and mathematical representation. Section IV provides a detailed explanation of the experimental results, while also describing the simulation setup, comparison analysis, and research summary. Section V concludes this research. Section VI presents the future works in IoT-enabled SDWSNs. Section VI provides a link for the dataset, the main project code, and a simulation video for this research.

## II. LITERATURE REVIEW

This section explains the concepts and research gaps of the existing works. The research concept, algorithm, and research gaps are listed in Table 1. And in Table 2, more comparisons of the proposed model and the existing works are conducted with respect to some of the topics mentioned in our work.

Rajan et al. [27] proposed an intrusion detection system for a cloud-based WSN-IoT network. This research used two levels of security for detecting network attacks. First, authentication was proposed based on the trust level of the nodes. The private keys for the routing nodes are validated by string values. Here, the binomial algebraic expansion method was used for generating false IDs for nodes. The second level of security was used for verifying the false IDs of the nodes using context-free grammar constraints. If the string values of the nodes were matched with the already generated map, then it would be considered a trusted node, otherwise, it would be known as a malicious user.

Ramadan [28] proposed a specific method for intrusion detection and prevention in the smart city-based WSN environment. Initially, the network was divided into multiple clusters, with the best cluster head (CH) chosen for each. The messages were sent to the CH through the optimal route. For that purpose, multipaths were detected based on modified AODV protocols, in which the messages were sent to the sink nodes by generating control messages that were verified by the sink nodes, thereby determining whether the message was altered or not. If the sink node detects a malicious message, the CH notifies the network about intrusions to prevent it from happening again. Another threshold-based method was used in this research for identifying intrusions in the network. In this case, the sink nodes performed intrusion detection, resulting in high complexity and overhead messages in a real-time environment because the sink node cannot process large amounts of data in a given time.

Goyat et al. [29] proposed a secure localization method using blockchain for the WSN environment. The proposed work includes two methods: the first is trust calculation, and the second is block generation. Here, both direct and indirect trust were calculated for sensor nodes. The trust calculation takes into account the following metrics: energy, reputation, and integrity, with trust values being evaluated based on weight factors. After that, the blockchain was generated using a proof of work consensus algorithm. Every block was generated by validating its legitimacy. Finally, localization was performed based on the highest trust values. The experimental results demonstrate that the proposed work achieved better performance in terms of true positive rate, accuracy, and false positive rate. Here, blockchain was used for increasing security; however, the traditional structure of blockchain leads to less scalability and higher energy consumption due to its linear structure.

An ensemble optimization technique was proposed for optimal routing in WSN [30]. The ensemble optimization includes genetic and particle swarm optimization (PSO),

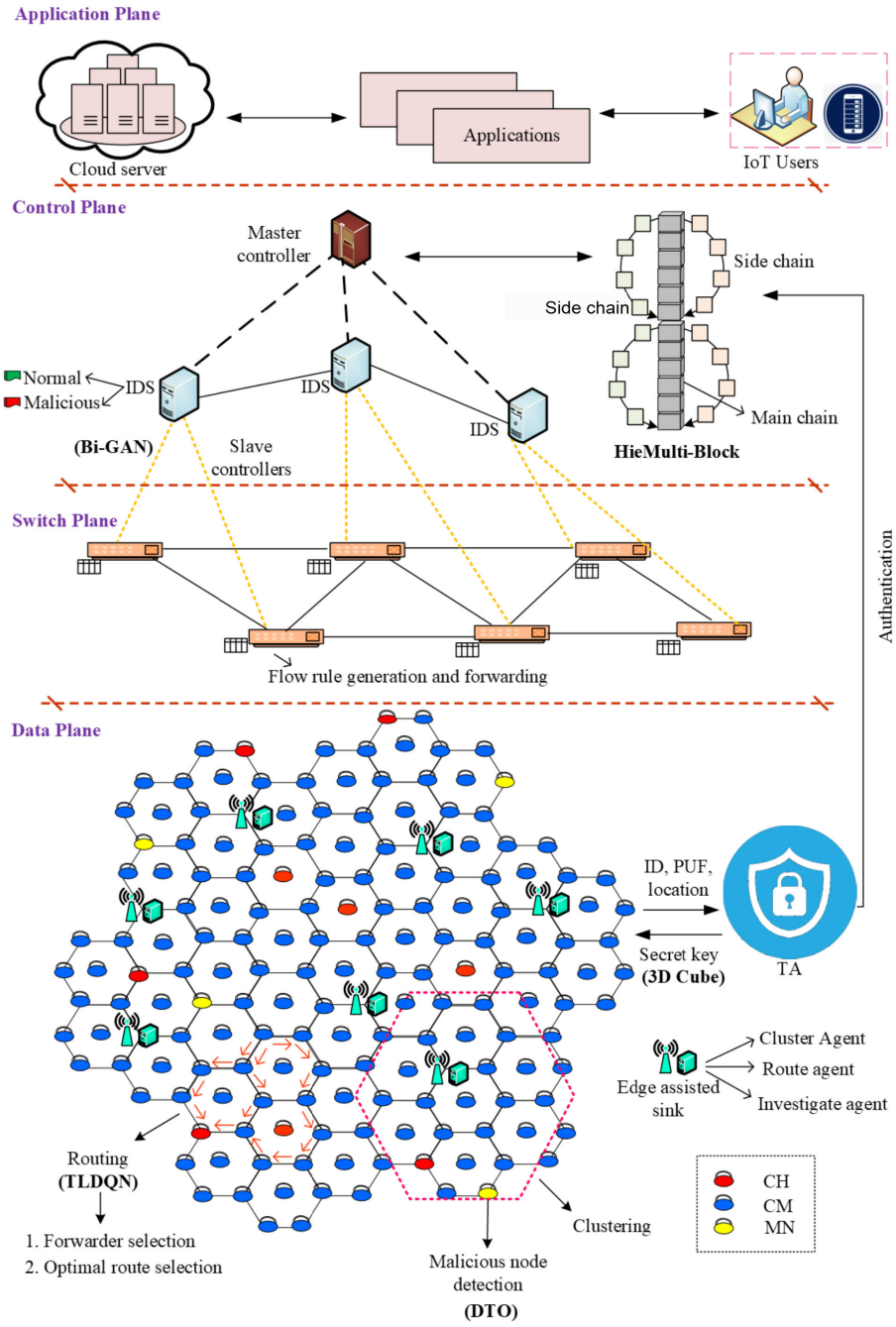



**FIGURE 1.** Architecture of the proposed system model.

which performs optimal control node selection by considering distance and energy parameters. This research considered both single and multiple sink nodes for communication. The control nodes were elected from among the nodes present in the network. The fitness values of the algorithm were evaluated in order to select the best control nodes for routing. Finally, the data packets were sent to the control server using optimal routing. The simulation result demonstrated that the proposed work achieved good performance in terms of energy and fitness. Here, optimal routing was performed based on a genetic mutation-based PSO algorithm; however, it did not consider the security metrics for routing that lead to high-security threats and high information loss.

Haseeb et al. [31], proposed a secure and energy-based routing using an optimization algorithm in WSN. This paper includes two processes: the first is an artificial intelligence-based secure routing, and the second is authentic and secure routing using an encryption algorithm. Initially, the heuristic algorithm calculated the weight values for selecting the next forwarder based on direction. Then the

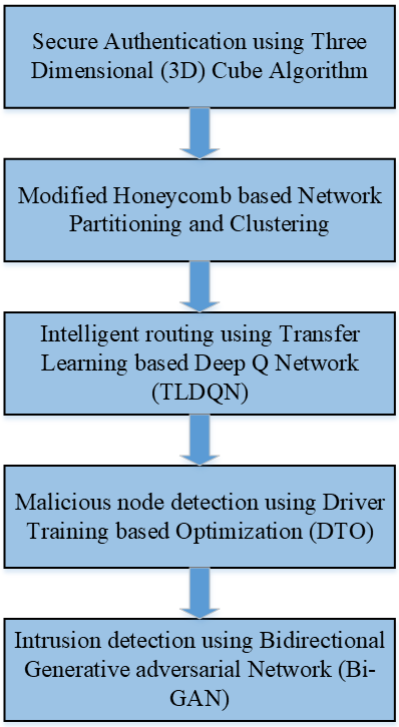



**FIGURE 2.** Flowchart of proposed model.

data connectivity was evaluated for the selected forwarder. Encryption was applied to the data to ensure security. Finally, the route maintenance process was performed to reduce the link failures. The experimental results show that the proposed work achieved better performance in terms of throughput, packet delivery ratio, energy consumption, end-to-end delay, and communication overhead. Here, secure and energy-based routing was performed; however, the limitation of hop counts was not considered, which increased the latency and energy consumption during data transmission.

Liu et al. [32] proposed a machine learning-based intrusion detection system in WSN. This research proposed edge-based intrusion detection using an improved k-nearest neighbors (k-NN) algorithm. The Euclidean distance-based weight values were used for classifying the normal and intrusion nodes in the network. The WSN environment has a resource constraint problem, which is overcome by integrating edge computing into the WSN, which detects intrusions using the proposed algorithm. The performance of the proposed work was evaluated in terms of accuracy, true negative rate, and true positive rate compared to existing approaches.

A new approach was proposed for performing an intrusion detection system in a WSN environment [33]. The proposed work included three phases: the data collection phase, the detection phase, and the response phase. Initially, the data was collected from the environment. The analyzer analyzed the collected data based on its traffic information. The sequence backward-based feature selection method was then proposed to reduce the traffic's irrelevant features. Based on the selected feature, the LightGBM algorithm was used to perform attack detection, which detects and classifies the packets into four classes: normal, gray hole, blackhole, and flooding attacks. Here, intrusion detection was performed based on LighGBM, however, it provides more complex trees that increase overfitting, which reduces the performance of intrusion detection.

A machine learning technique was proposed for detecting DDoS attacks in an SDN environment [34]. The proposed work includes two planes, namely the data plane and the control plane. In the data plane, the detection trigger method was deployed to analyze the traffic patterns of the node through switches. For that purpose, the flow features were extracted from the traffic patterns by calculating the rate of traffic. If it exceeds the threshold, then it will be known as an abnormal flow; otherwise, it is a normal flow. Here, k-NN and K-means algorithms were used for feature extraction and DDoS detection. A single SDN controller was used for DDoS attack detection, which led to high control traffic overhead during the large amount of data processing that increased latency in DDoS attack detection.

Lakshmanna et al. [35] proposed a metaheuristic algorithm for energy-efficient clustering and routing in WSN. Initially, the nodes are randomly placed for communication and information collection. After that, the improved Archimedes optimization algorithm was proposed for performing clustering by considering energy, distance, and node degree. After completing clustering, routing was initialized using a teaching-based optimization algorithm that performs multi-hop routing to select the optimal route by considering distance and energy. The simulation result demonstrates that the proposed work achieved better performance in terms of lifetime, energy consumption, latency, and packet delivery ratio. Here, nodes are randomly placed in an environment that increases complexity and inefficiency in network management due to its unstable nature. Additionally, routing was performed based on an optimization algorithm without considering any security metrics that increase security threats.

An authentication information exchange method was proposed for detecting attacks in WSN-based IoT applications [36]. The main aim of this research is to mitigate attacks in the IoT environment. The internal joint defense method is proposed to verify whether the sensor node is available or not. Additionally, it tracked the nature of the sensor and was classified into three classes, such as normal, compromised, and partially compromised. If the mechanism detects the attacks, then it initiates a prevention mechanism. If the node was normal, then the information was exchanged between the sensor nodes. The experimental result demonstrates that the proposed work achieved better security by performing authentication-based information exchange. Here, authentication was evaluated based on the nature of the node, which leads to poor security because the attacker can easily compromise the node and act as a legitimate node, leading to high information loss.

Bhatt et al. [37] proposed an optimization algorithm for detecting capture attacks in WSN. The proposed work analyzed the optimal sensor nodes using the fruit firefly optimization algorithm. Initially, a key route matrix was generated to define the relationship between the keys and routes. The number of keys was evaluated based on the key number matrix. After that, the vertex route matrix was generated to evaluate the routes in the WSN. Here, capture attacks were detected by calculating the energy consumption. The simulation result demonstrates that the proposed work achieved better performance in capture attack detection compared to existing works.

Khan et al. [38] proposed a method for detecting abnormal traffic in WSN. Initially, the data packets were captured from the sensor nodes with the properties of source and route information, size of packets, and arrival time. The features are then extracted from the data packets in order to detect DDoS attacks. Based on the extracted features, this research performed attack detection. Finally, the proposed Bayesian model classified the traffic into normal traffic or DDoS traffic. The simulation results show that the proposed work achieved better performance in terms of attack detection rate, false negative rate, and false positive rate. Here, all the nodes are considered legitimate and allowed to perform data transmission that increases data traffic, which increases complexity and reduces the accuracy of traffic prediction.

Jurado-Lasso et al. [39] proposed a new routing mechanism for reducing the control overhead in the WSN environment. The proposed routing protocol reduced energy consumption by selecting an optimal path with the least amount of energy, increasing network reliability and lifetime. Initially, the algorithm generated a tree structure for sorting the information of sensor nodes, and then it selected the shortest path from that tree structure. To reduce control overhead, this research maintained the neighbor status and calculated the checksum for all the routes in the routing table. The experimental result demonstrated that the proposed work achieved better performance in terms of lifetime, packet delivery ratio, and control overhead.

Nguyen et al. [40] proposed a secure and energy-based clustering using the red deer algorithm for the WSN environment. Initially, nodes were randomly placed in the network and data from neighbor nodes was collected for clustering. After that, CH was selected by evaluating the fitness function by considering the distance, node density, and energy. Here, secure data transmission was performed using blockchain. The experimental results show that the proposed work achieved better performance in terms of energy, throughput, lifetime, and packet delivery ratio. Here, the red deer algorithm was proposed for clustering; however, it leads to high latency and energy consumption due to its low convergence.

Theodorou et al. [41] provided a solution for the IoT environment using SDN architecture. The proposed work includes three planes, namely the infrastructure plane, the control plane, and the application plane. Initially, mobility was addressed by considering mobility features that increase the stability of the network. For that purpose, this research generated policies for topology discovery, routing, and flow rule establishment. After that, the mobility detector detects the mobile nodes using the k-means algorithm. The performance of this research was evaluated in terms of control overhead and packet delivery ratio.

Bhayo et al. [42] proposed a strategy for detecting DDoS assaults in an SDN-WISE IoT controller, which used machine learning (ML) as its foundation. They have included a detection module that is based on integrating ML into the controller, and they have created a testbed setting to imitate the creation of DDoS attack traffic. A logging method that has been introduced to the SDN-WISE controller is responsible for capturing the traffic. This system sends network logs into a log file that is then pre-processed and transformed into a dataset. The ML DDoS identification part of the SDN-WISE controller uses the Naive Bayes (NB), Decision Tree (DT), and Support Vector Machine (SVM) classification methods to properly sort SDN-IoT network packets.

Siddiqui et al. [43] investigated published research on SDN-based architectures to handle IoT management concerns in the fields of failure tolerance, energy administration, scalability, load management, and safety-related service provisioning inside IoT networks. It presents a complete assessment of this research. By using Software Defined Network (SDN) and blockchain technology in an IoT environment, the study in [44] offers a monitored and transparent access control policy management platform that hinders the propagation of forged regulations and addresses the complexity of policy administration, forgeries, propagation, observing access control policies, automation, and central administration of IoT nodes. A reliable solution for IoT environment security is provided by the combination of SDN and blockchain. They include an effective proof of concept that proves the scalability of the suggested remedy, along with an innovative, scalable approach for building immutable, accessible, adaptable, and automatic access control rules for IoT devices.

Alotaibi et al. [45] proposed a method for routing in IoT-based WSN networks, involving multipath routing and secure transmission. The research faced challenges such as: not providing an optimal path due to limited feature consideration, leading to increase latency and congestion; using an optimization algorithm causing high latency and energy consumption; allowing all nodes to transmit data, increasing security breaches; and not providing full security due to centralized storage and focusing only on routing attacks, leaving the network vulnerable to intrusions.

A trusted-aware routing method using deep learning for WSN environments was proposed in [46]. The aim is to create a secure route, but the random placement of sensor nodes affects network performance and increases packet loss. The routing method enhances security but results in high

**TABLE 1. Summary of literature survey.**

| References | Concept | Algorithm/Method | Limitations |
|---|---|---|---|
| Rajan *et al.* [27] | Anonymous intrusion detection system for providing security in cloud-based WSN-IoT network | Binomial algebraic expansion method | Low security |
| Ramadan [28] | Intrusion detection and prevention were performed for smart city-based WSN environment | Threshold-based method | High complexity and high overhead communication |
| Goyat *et al.* [29] | Secure localization method was proposed for the WSN environment using blockchain | Proof of work consensus algorithm | Less scalable and high energy consumption |
| Ramteke *et al.* [30] | Ensemble optimization-based optimal routing was performed in the WSN environment | Genetic and particle swarm optimization | High information loss |
| Haseeb *et al.* [31] | Optimization algorithm was proposed for performing secure and energy-based routing in WSN | Artificial intelligence-based graph heuristic algorithm | High latency and high energy consumption |
| Liu *et al.* [32] | Machine learning-based intrusion detection system was proposed for the WSN environment | Improved k-NN algorithm | Low detection accuracy |
| Jiang *et al.* [33] | Machine learning-based intrusion detection system for smart environment | Lightweight gradient boosting algorithm | Low detection accuracy |
| Tan *et al.* [34] | Machine learning-based DDoS attack detection in SDN environment | k-NN and k-means clustering algorithm | High latency and high overhead communication |
| Lakshmanna *et al.* [35] | Energy-efficient clustering and routing were performed by using a metaheuristic algorithm | Archimedes optimization algorithm | Poor security and high complexity |
| Yang *et al.* [36] | Secure information exchange-based attack mitigation in WSN-based IoT | Authentication information exchange | High information loss |
| Bhatt *et al.* [37] | Optimization-based attack detection in the WSN environment | Fruit firefly optimization | High latency |
| Khan *et al.* [38] | Abnormal traffic was detected by the multilevel probabilistic model in WSN | Bayesian model | Low detection accuracy |
| Jurado-Lasso *et al.* [39] | Routing algorithm was proposed to reduce control overhead in WSN | Energy-aware routing algorithm | Low throughput |
| Nguyen *et al.* [40] | Secure and energy-based clustering using optimization algorithm for WSN | Red deer algorithm | High latency and high energy consumption |
| Theodorou *et al.* [41] | SDN is incorporated with IoT for providing a solution for the mobile IoT environment | K-means algorithm | High complexity |
| Alotaibi *et al.* [45] | A novel method for routing in IoT-based WSN networks | Optimization algorithm for multi-path selection | High latency and high energy consumption |
| El-Moghith *et al.* [46] | A trusted-aware routing method using deep learning for WSN environments | Dijkstra and blockchain-based routing algorithms, and Q-Learning backpressure algorithm (QL-BP) | High latency, low throughput, and inefficient management |
| Younus *et al.* [47] | A reinforcement learning method for optimal routing in SDWSN environments | Q learning algorithm | High latency and high energy consumption |
| Srividya *et al.* [48] | A machine learning algorithm for intrusion detection in a WSN environment | Decision tree algorithm | High complexity and high energy consumption |
| Sixu *et al.* [49] | A clustering and routing mechanism using a hybrid optimization algorithm in a WSN environment | ABC-based traversal path algorithm and PSO-based cluster routing algorithm | High latency and high energy consumption |

**TABLE 2. Comparison of our proposed model with other existing works.**

| Topics mentioned | [27] | [28] | [29] | [32] | [34] | [36] | [38] | [39] | [46] | [47] | [48] | [49] | [59] | Our proposed model |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IDS | ✔ | ✔ | ✗ | ✔ | ✔ | ✗ | ✔ | ✗ | ✗ | ✗ | ✔ | ✗ | ✗ | ✔ |
| Blockchain | ✗ | ✗ | ✔ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ | ✗ | ✗ | ✗ | ✗ | ✔ |
| Security in IoT-based WSN | ✗ | ✔ | ✗ | ✗ | ✗ | ✔ | ✗ | ✔ | ✗ | ✗ | ✗ | ✔ | ✗ | ✔ |
| Clustering in SDWSN | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ | ✗ | ✔ |
| Honeycomb structure | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ |
| RL-based intelligent routing | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ | ✗ | ✗ | ✗ | ✔ |
| Secure authentication | ✔ | ✔ | ✔ | ✗ | ✗ | ✔ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ | ✔ |
| Intrusion detection via machine learning | ✗ | ✗ | ✗ | ✔ | ✔ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ | ✗ | ✔ |
| Intrusion detection via deep learning | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✔ | ✗ | ✗ | ✗ | ✔ |

✔- Included
✗ - Not Included

latency, reduced throughput, and inefficient management. Storing information in a single linear blockchain also causes high latency, increased energy consumption, and limited scalability, adding complexity to secure routing in real-time environments.

A reinforcement learning method for optimal routing in SDWSN environments was proposed in [47], using the Q learning algorithm. The study had two planes: the data plane and the control plane. The main challenges were: increased energy consumption and reduced throughput due

to random sensor node localization; insufficient criteria for optimal routing selection, leading to packet loss; a single centralized controller causing complexity and inefficiency; and routing tables maintained without security, increasing threats and information loss. Additionally, the Q learning algorithm caused high latency and energy consumption due to its low convergence and many iterations needed for optimal routing selection.

The study in [48] proposed a machine learning algorithm for intrusion detection in WSN using clustering and routing processes. However, it faced several challenges: legitimate nodes were identified based on trust values, affecting throughput and packet delivery ratio; trust values were insufficient for accurate attack detection and could be compromised; the decision tree algorithm detected intrusions but lacked prevention, leading to information loss and degraded performance; and data collection and processing by sink nodes or base stations which led to increased complexity and energy consumption, and also reduced network lifetime.

A clustering and routing mechanism using a hybrid optimization algorithm in a WSN environment was proposed in [49]. The research focuses on two processes: cluster routing and traversal path selection. The algorithm used in this study considers cluster head (CH) selection and base station (BS) sojourn locations, aiming to reduce energy consumption and improve communication efficiency. However, there are some challenges identified. Firstly, the routing process performed by controller nodes does not take into account trust values or security metrics, leading to high vulnerabilities in the WSN environment and low throughput. Secondly, the research employs the ABC-based traversal path algorithm to find the shortest path for the mobile BS, but it faces the discrete problem of the traveling salesman problem (TSP). Lastly, the base station's placement in the controller plane results in high latency and energy consumption, impacting the sensor nodes' lifetime.

### A. RESEARCH SOLUTIONS

In our work, the 3D cube algorithm is used for secure authentication, where all of the sensor nodes register their parameters such as ID, physical unclonable functions (PUF), location, and random number to the trusted authority (TA) to make the system more secure. In our work, the network is partitioned based on a modified honeycomb structure that places the sensor nodes on the edges of the hexagon, which increases the performance of network management. In our work, RL-based intelligent routing is performed, which selects the optimal path with a minimum amount of time by performing next forwarder selection and optimal path selection by considering trust values and other routing metrics. Furthermore, secure routing and hybrid intrusion detection systems are used to detect both routing attacks and intrusions in the network, increasing the security and throughput of the SDWSN environment. In our work, an intrusion is detected based on the bidirectional generative adversarial network (Bi-GAN) algorithm by considering both packet and flow-based features, which increases security; in addition, optimal delegator-based intrusion prevention is performed in this research, which protects the network from the attackers efficiently. Edge-based sink nodes collect the data from the sensor nodes and send it to the controller nodes, which reduces complexity and energy consumption and increases the network lifetime. In this research, hierarchical-based multiple blockchains (HieMulti-Block) are used, which reduces energy consumption and increases scalability. All information is stored in HieMulti-Block, which makes security better because blockchain is an unchangeable ledger that cannot be hacked or changed by attackers.

## III. SYSTEM MODEL

The foremost aim of this research is to detect intrusions with less energy consumption and latency in the SDWSN environment. The proposed work includes four planes, namely the application plane, control plane, switch plane, and data plane. Users can access information on the application plane via IoT devices. The control plane includes two types of controllers: the primary controller and secondary controllers. The switch plane includes a number of switches for generating flow tables. The data plane includes several sensor nodes with edge-assisted sink nodes. The Edge server includes three types of agents: cluster agent, routing agent, and investigate agent. Table 3 depicts the goals of the proposed work, and Fig. 1 represents the architecture of the system model. This research includes four main processes, which are:

- Secure authentication
- Modified honeycomb based network partitioning and clustering
- RL-based intelligent routing
- Hybrid intrusion detection system

### A. SECURE AUTHENTICATION

Initially, the sensor nodes must register their parameters with the trusted authority (TA) to ensure their legitimacy. For the sensor node, we have considered parameters like ID, physical unclonable functions (PUF) [50], location, and random information or number (the shuffle number of movements used to construct a private key in the 3D cube algorithm and is obtained by running *random()* without particular restrictions rather than utilizing just the specified number of movements) provided by the authenticated node during registration. For the IoT user, we have to consider parameters like ID, PUF, MAC, and location, which are registered to the TA and stored in the HieMulti-Block in a hashed manner. After completing registration, the TA provides a private key with the aid of the **three-dimensional cube (3D cube) algorithm** [51], which creates a private key based on symmetric key encryption, in which the private key is generated based on the deep neural network without sharing a pre-shared key that increases the security and confidentiality. Fig. 3 depicts the architecture of authentication using the 3D cube algorithm. On the receiver side, the 3D cube pattern is solved to generate the private key. For solving the cube pattern,

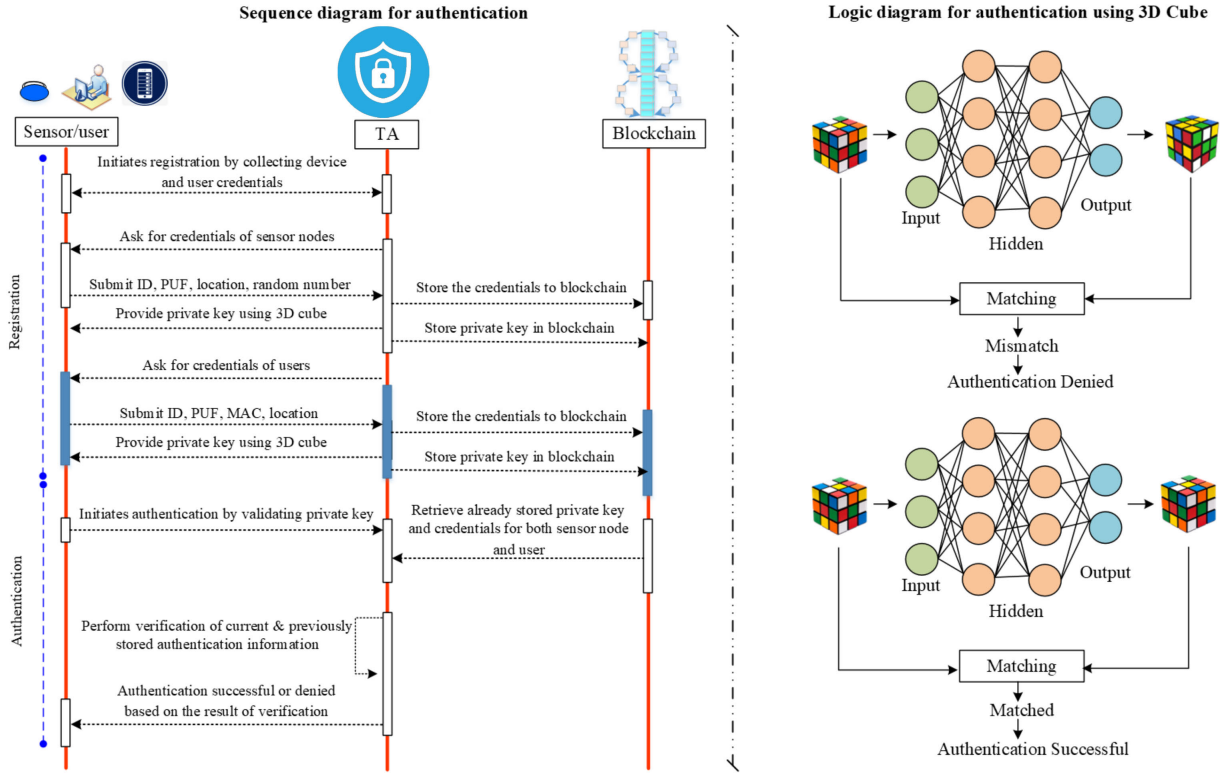



**FIGURE 3.** Authentication using 3D cube algorithm.

**TABLE 3.** Goals of the proposed model.

| Process | Algorithms | Goals |
|---|---|---|
| Secure Authentication | Three-dimensional cube | • Improved security<br>• Avoid external attacks |
| Modified honeycomb-based network partitioning clustering | —- | • Improved energy efficiency<br>• Improved network management |
| RL-based intelligent routing | Transfer learning-based deep Q learning | • Improved throughput<br>• Reduce packet loss |
| Hybrid intrusion detection and prevention system | Driver training-based optimization Bidirectional GAN | • High detection accuracy<br>• Improved security |

a deep neural network is implemented, which is an artificial intelligence technique known as the ''deep cube algorithm'', which provides intelligent cube patterns with high speed. For generating the private key, the proposed algorithm verifies the cube pattern is matched with the coordinates of the array, whereas the cube shuffling order is defined as an arrangement for solving the cube patterns and generating the private key by mapping the resultant by performing an XOR process on the direct value for every move in the cube. A WSN requires a trusted authority capable of authenticating devices and transactions. This is possible with private keys, which are cryptographic keys that the user keeps concealed. A trusted authority is able to issue private keys to devices and use them for transaction authentication. The proposed deep cube algorithm solves any cube pattern within 30 movements of cube arrangements.

To provide a general overview of the steps involved in solving, ordering, and key generation in the 3D cube algorithm, the process is as follows:

- Cube pattern solving: Given an initial cube pattern, the solver attempts to find a series of moves or transformations that will lead to a desired target pattern. This can involve applying predefined methods to manipulate the cube's elements (e.g., rotating specific layers) until the target pattern is reached.
- Cube pattern ordering: Once the cube patterns are solved, they can be ordered based on a specific criterion. Ordering can be done for various purposes, such as optimizing the sequence of moves required to reach a target pattern or analyzing the structure of the cube.
- Key generation: From the extracted arrangement or ordered cube patterns, a private key can be generated. The private key is typically a unique identifier or

secret value derived from the cube pattern arrangement. The process of generating the private key can involve combining certain elements or attributes of the cube patterns and applying cryptographic algorithms such as hash functions to ensure uniqueness and security.

- Hashing: The generated private key is then hashed using a secure hash function. Hashing is a one-way function that transforms the private key into a fixed-size output, known as the hash value or digest. The purpose of hashing is to ensure the privacy and integrity of the private key by generating a unique and irreversible representation of it. The hash functions for cryptographic purposes, such as generating private keys or ensuring the integrity of data, include SHA-256 (secure hash algorithm 256-bit). It produces a 256-bit hash value and is widely used for cryptographic purposes.

The hashed private key and arrangement number are sent to the sender side. If the sender indicates that the generated secret key matches, the key generation process should be terminated. Otherwise, proceed to the key matching process from the first stage. Authentication should be performed once the secret key generation process is complete. The final pattern of the cube is generated based on the Monte Carlo tree search (MCTS) algorithm. The search tree process begins with the initial state and continues until it reaches the tree's termination node.

Every state is linked with memory, which stores many variables such as the count of move $b$ taken at state ($\varepsilon_{St}(b)$), the maximum value of move $b$ at state ($\gamma_{St}(b)$), the present virtual loss (the loss that results from not searching the same tree search state repeatedly prevents the asymmetric task from pursuing the same route) of move $b$ at state ($\delta_{St}(b)$), and the probability of a predicted action from state ($\alpha_{St}(b)$). The MCTS policy solves the cube pattern and controls the move for every step, which is explained as:

$$O_t = \text{Argmax}_b \, V_{S_t}(b) + R_{S_t}(b) \tag{1}$$

$$V_{St}(b) = dP_{St}(b)\sqrt{\Sigma_b \frac{\varepsilon_{St}(b')}{1+\varepsilon_{St}(b)}} \tag{2}$$

$$R_{St}(b) = \gamma_{St}(b) + \delta_{St}(b) \tag{3}$$

$$\delta_{St}(O_t) \leftarrow \delta_{St}(O_t) + w \tag{4}$$

The highest optimal value at child state ($O_t$) is calculated based on $V_{St}(b)$ and $R_{St}(b)$ with respect to time $t$. The possibility of previous action is calculated based on the number of move $b$ in the cube shuffling state, and $R_{St}(b)$ is calculated based on considering virtual loss with respect to the highest primary value at state $St$ and the value of the virtual loss is updated steadily by using $w$ parameter. Once the searching process is reached the lead node of the tree, the minimum elements $St$ and $\{O(St, b), \forall b \in O\}$ elements are added to the search tree for expanding the state of the tree. After completing tree expansion, the $St'$ memory of every child node is initialized as:

$$\varepsilon_{St}(b) \rightarrow \varepsilon_{St'}(\cdot) = 0 \tag{5}$$

$$\gamma_{St}(b) \rightarrow \gamma_{St'}(\cdot) = 0 \tag{6}$$

$$\delta_{St}(b) \rightarrow \delta_{St'}(\cdot) = 0 \tag{7}$$

$$\rho_{St}(b) \rightarrow \rho_{St'}(\cdot) = \rho_{St'} \tag{8}$$

where $\rho_{St'}$ represents the network function $F_\vartheta(St')$ policy output. The cube pattern is solved by calculating the highest policy value, and the optimal value of child state, which is expressed as:

$$(w_{St_{\vec{\sigma}}}, \sigma_{St_{\vec{\sigma}}}) = F_\vartheta(St') \tag{9}$$

The memory update occurring over time in $0 \leq t \leq \sigma$ can be calculated using:

$$\gamma_{St}(O_t) \leftarrow \max(\gamma_{St}(O_t), w_{St_{\vec{\sigma}}}) \tag{10}$$

$$\varepsilon_{St}(O_t) \leftarrow \varepsilon_{St}(O_t) + 1 \tag{11}$$

$$\delta_{St}(O_t) \leftarrow \delta_{St}(O_t) - w \tag{12}$$

This type of calculation helps to solve the cube patterns that provide the secret key by using the deep cube algorithm, which is trained by the deep neural network (DNN) algorithm, to solve the cube patterns and send them to the sender side. The DNN is a type of artificial neural network (ANN) that includes an input layer, a hidden layer, and output layers that provide optimal secret key generation. To protect against cyberattacks, the TA provides a private key with the aid of the 3D cube algorithm. Here, authentication is done by validating the secret key generated by the 3D cube algorithm. After completing authentication, clone attack detection is performed to increase network security. It is detected based on identifying the same ID presented in different locations. In this case, the random information or number is evaluated and provided by the authenticated node during registration. If it is valid, then the authenticated node is allowed to be present in the network with other authenticated nodes.

### B. MODIFIED HONEYCOMB-BASED NETWORK PARTITIONING AND CLUSTERING

The major issue in WSN is high energy consumption and latency during data transmission. To overcome these issues, we build a network based on a modified honeycomb structure, which increases both coverage and throughput. In this research, we proposed a 2-dimensional honeycomb structure, which includes a 2-dimensional hexagonal structure. In a traditional honeycomb structure, the sensor nodes are placed in the center of the hexagon, which covers a smaller number of nodes. In this research, we placed the sensor nodes on the edge and center of the hexagons, as seen in Fig. 4, which increases coverage with an increased number of nodes.

Every hexagonal cell has six neighboring cells in all directions. Every sensor node can communicate with all the neighboring sensors that are placed on the adjacent edges of the hexagon. Clustering begins after sensor placement and network construction are completed. The sensor nodes

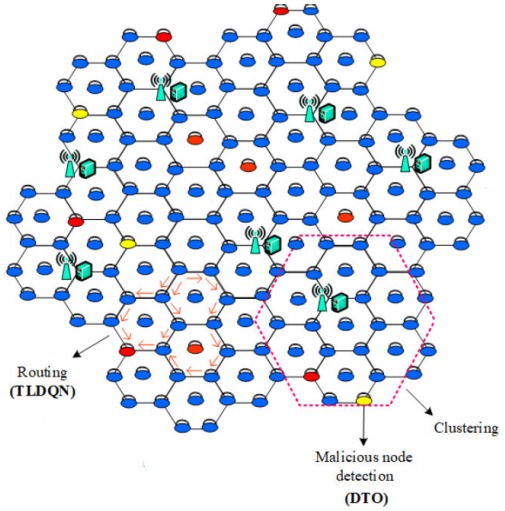



**FIGURE 4.** Sensor nodes' placement in a honeycomb structure.

are clustered based on a hexagonal structure based on a 1/7 fraction, which means each cluster includes seven hexagonal cells in the honeycomb. Here, clustering was done by a cluster agent that was deployed on the sink node. The total number of hexagons and deployed sensors are calculated as:

$$K_{sensor} = c + \sum_{i=0}^{n} (K - i) \tag{13}$$

where $K$ represents the count of hexagonal layers essential for coverage, $c = 7$, which is equal to the innermost layer where the sensor count is 7. The total number of hexagons in a single cluster is $7i^2$, and the number of clusters in the environment is calculated as:

$$Cl_i = \begin{cases} 7, & \text{if } i = 1 \\ 7(i-1), & \text{if } i \geq 2 \end{cases} \tag{14}$$

Here, 7 hexagons are grouped to form a single cluster, and the count of the sensor calculation of the cluster is defined as:

$$S_c = c \times 7 \tag{15}$$

After completing clustering, the cluster head (CH) is selected based on node centrality ($\mathbb{c}$), distance ($\mathbb{d}$), energy ($\mathbb{e}$), and trust ($\mathbb{t}$). The weight coefficients of the parameters are calculated for selecting the optimal CH; if the CH has a high weight value, then it is selected as the current CH. The CH selection is expressed as:

$$W_{CH} = \Sigma(w_1 \times \mathbb{c}) + (w_2 \times \mathbb{d}) + (w_3 \times \mathbb{e}) + (w_4 \times \mathbb{t}) \tag{16}$$

Each part of clustering has a weight coefficient $W_i(w_1, w_2, w_3, w_4)$ that shows how important it is to CH selection and residual energy.

In modified honeycomb-based network partitioning and clustering (MHPC), the weight coefficients $(w_1, w_2, w_3, w_4)$ for CH selection are typically determined based on various factors and considerations (such as network lifetime, communication efficiency, QoS, etc.), depending on the specific implementation and objectives of the network.

The weight coefficients are usually assigned based on their relative importance in the CH selection process. These coefficients are used to calculate a composite metric or score for each sensor node, and the nodes with higher scores are chosen as cluster heads. The specific calculation of the composite metric may vary, but it typically involves combining multiple parameters to evaluate the suitability of a node for being a cluster head.

Here is a general overview of the weight coefficients and their potential significance in the CH selection process:

- $w_1$ (remaining energy or battery power): This weight coefficient represents the importance of the energy level of a node. Nodes with higher energy levels are typically preferred as cluster heads to ensure longer network lifetime. $w_1$ determines the relative significance of energy in the overall CH selection process.
- $w_2$ (distance to the base station): This coefficient indicates the importance of the proximity of a node to the base station or sink node. Nodes closer to the base station may be preferred as cluster heads to minimize energy consumption for data transmission. $w_2$ determines the relative significance of the distance in the CH selection process.
- $w_3$ (node degree or connectivity): This coefficient reflects the significance of node connectivity or the number of neighboring nodes. Nodes with higher connectivity may be more suitable as cluster heads to improve network coverage and routing efficiency. $w_3$ determines the relative importance of node connectivity in the CH selection process.
- $w_4$ (residual bandwidth): This weight coefficient represents the available bandwidth or capacity of a node for data transmission. Nodes with higher residual bandwidth may be preferred as cluster heads to handle increased traffic or data aggregation tasks. $w_4$ determines the relative importance of residual bandwidth in the CH selection process.

It is important to note that the specific values assigned to these weight coefficients can vary depending on the specific network deployment and requirements. The values can be determined through simulation studies, empirical analysis, or optimization techniques based on the desired network performance objectives, such as maximizing network lifetime, minimizing energy consumption, or improving data aggregation efficiency. With this method of network partitioning and clustering, the network can be managed well and use less energy.

### C. RL BASED INTELLIGENT ROUTING

Routing is an important process in SDWSN to reduce network congestion and increase throughput. It is performed for data transmission between CH and cluster member (CM) by the

route agent using transfer learning-based deep $Q$ network (TLDQN) [52], [53], which is under reinforcement learning, as shown in Fig. 5, and its pseudocode is given in Algorithm 1. Traditional reinforcement learning faces a low learning rate problem due to the need to learn several network parameters. To overcome this issue, we have applied transfer learning to the RL to improve learning efficiency and speed. Here, we perform two processes for intelligent routing: first, the optimal next forwarder is selected based on *trust, distance, link quality, and energy*; second, the optimal route is selected based on *the number of hops, link stability, packet delivery ratio, and throughput*. This routing process was performed by the routing agent deployed on the edge server. The state of TLDQN, like other deep $Q$-network variants, includes parameters that are learned through the training process. These parameters capture the knowledge and policies learned by the network. In this research, the actions of TLDQN are to first perform optimal forwarder selection and then optimal route selection by considering optimal next forwarders. The benefit of using RL TLDQN is an increased throughput and packet delivery ratio. The reward function is an essential component of reinforcement learning algorithms, including TLDQN. It represents the feedback or evaluation of the network's state after an action is taken. The reward guides the learning process, allowing the network to learn from the consequences of its actions and adjust its policy accordingly.

In the TLDQN algorithm, the $Q$-value represents the estimated value of taking a particular action in a specific state. The $Q$-value is a crucial component in reinforcement learning algorithms, particularly in $Q$-learning and its deep learning variants.

In the context of TLDQN, the $Q$-value is typically represented by a neural network, where the input is the state of the environment and the output is the $Q$-value for each possible action in that state. The $Q$-value function can be denoted as $Q(s, a)$, where $s$ represents the state and $a$ represents the action.

During the training process of TLDQN, the $Q$-values are learned and updated based on the observed rewards and the estimated future rewards. The goal is to approximate the optimal $Q$-values for each state-action pair, which indicate the expected return or cumulative reward that can be achieved by taking a specific action in a given state. The $Q$-values are updated iteratively using the *Bellman* equation or its variants. This equation defines how the $Q$-values are updated based on observed rewards and the estimated future rewards. The *Bellman* equation [54] is defined as:

$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a')) \tag{17}$$

where:

- $Q(s, a)$ represents the $Q$-value of taking action $a$ in state $s$.
- $\alpha$ (alpha) is the learning rate that determines the weight given to new information compared to the existing $Q$-values.
- $r$ is the observed reward received after taking action $a$ in state $s$.
- $\gamma$ (gamma) is the discount factor that determines the importance of future rewards. It represents the weightage given to future rewards compared to immediate rewards.
- $s'$ represents the next state observed after taking action $a$ in state $s$.
- $a'$ represents the action selected in the next state when updating the $Q$-value for the current state-action pair in the *Bellman* equation.
- $\max_{a'} Q(s', a')$ is the maximum $Q$-value among all possible actions $a$ in the next state $s$.

The proposed TLDQN maintains an experience replay for storing and extracting the batch samples from the replay buffer database. TLDQN, similar to other DQN (deep $Q$-network) variants, utilizes an experience replay mechanism. It stores past experiences (state-action-reward-next state tuples) in a replay buffer database and samples batches of experiences randomly for training. Experience replay helps to break the correlation between consecutive samples and stabilize the learning process.

The DQN includes two neural networks that contribute to training stability. One network, known as the $Q$-network or the online network, is responsible for estimating $Q$-values for state-action pairs. The other network, called the target network, is used to generate target $Q$-values for updating the $Q$-network. During training, the $Q$-network is periodically updated using a loss value calculated based on the difference between the estimated $Q$-values and the target $Q$-values, which guides the adjustment of the neural network's weights.

Here is a brief explanation of the statement above:

- $Q$-network (Online network): The $Q$-network is a neural network that approximates the $Q$-values for each state-action pair. It takes the current state of the environment as input and produces estimated $Q$-values for all possible actions in that state. The $Q$-network is updated during the training process to improve its accuracy in estimating $Q$-values.
- Target network: The target network is another neural network used to generate target $Q$-values. It is a copy of the $Q$-network that provides the desired $Q$-values used as targets during the training process. The target network is kept fixed or updated less frequently compared to the $Q$-network, which helps stabilize the learning process.
- Loss value: The difference between the estimated $Q$-values (from the $Q$-network) and the target $Q$-values (from the target network) is used to calculate a loss value. The loss value quantifies the discrepancy between the predicted $Q$-values and the desired $Q$-values. By minimizing this loss, the $Q$-network learns to better approximate the optimal $Q$-values.

During training, the $Q$-network is updated using *back-propagation* or backward propagation of errors (it is a key algorithm used in training artificial neural networks. It is a method for calculating the gradient of the loss function with

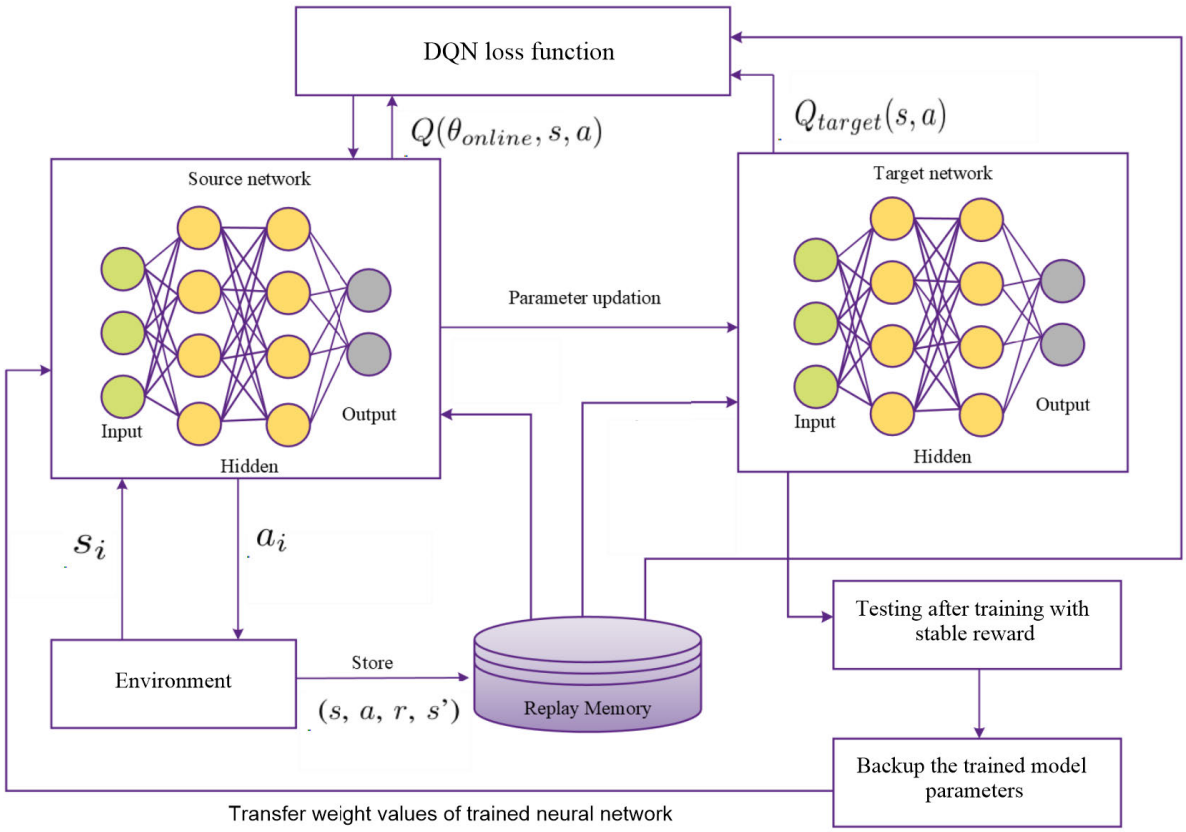



**FIGURE 5.** Architecture of TLDQN.

respect to the weights and biases of the neural network) and gradient descent to minimize the loss between the estimated and target $Q$-values. The target $Q$-values are generated using the *Bellman* equation (17), which incorporates the rewards and future $Q$-values to update the $Q$-values iteratively.

In the TLDQN algorithm, The loss function measures the discrepancy between the predicted $Q$-values and the target $Q$-values. It computes the squared difference between the predicted and target $Q$-values for each state-action pair and then averages the squared differences over the entire batch of training samples.

The objective of the training process is to minimize this loss function by adjusting the neural network's parameters (weights and biases) through backpropagation and optimization algorithms. Minimizing the loss function helps the network improve the accuracy of its $Q$-value estimates, leading to better performance and decision-making in reinforcement learning tasks.

The formula for calculating the loss function, specifically the mean squared error (MSE) loss function, is defined as:

$$\text{Loss} = \frac{1}{N}\sum_{i=1}^{N}(Q_{\text{predicted}}(s_i, a_i) - Q_{\text{target}}(s_i, a_i))^2 \quad (18)$$

where:

- $Q_{\text{predicted}}(s_i, a_i)$ represents the predicted $Q$-value for state $s_i$ and action $a_i$ based on the current neural network's output.
- $Q_{\text{target}}(s_i, a_i)$ represents the target $Q$-value for state $s_i$ and action $a_i$, which is usually calculated using (17) or its variants.
- $N$ represents the total number of state-action pairs in the training batch. The division by $N$ in the loss function formula ensures that the loss is normalized by the number of samples in the batch. This normalization allows for a fair comparison of loss values across different batch sizes.

In DQN, the stochastic gradient descent (SGD) method is commonly used to update the weights and reduce the loss function. The updated weight values are obtained through the iterative adjustment of the weights based on the gradients of the loss function with respect to the network's parameters, multiplied by the learning rate. The learning rate determines the step size in each iteration, controlling the rate at which the weights are updated.

The SGD update equation is defined as:

$$W_{\text{new}} = W_{\text{old}} - \eta \cdot \nabla L \quad (19)$$

where:

- $W_{\text{new}}$ represents the updated weights.
- $W_{\text{old}}$ represents the current weights.

- $\eta$ represents the learning rate (step size).
- $\nabla L$ represents the gradient of the loss function with respect to the weights.

The gradient of the loss function $\nabla L$ with respect to the weights is computed using *backpropagation*. The specific equations depend on the architecture of the neural network and the loss function used. For the mean squared error (MSE) loss, the gradient can be calculated as:

$$\nabla L = \frac{1}{N}\sum_{i=1}^{N} 2 \cdot (Q_{\text{predicted}}(s_i, a_i) - Q_{\text{target}}(s_i, a_i)) \cdot \nabla Q_{\text{predicted}}(s_i, a_i) \quad (20)$$

where:

- $N$ is the total number of state-action pairs in the training batch.
- $Q_{\text{predicted}}(s_i, a_i)$ represents the predicted $Q$-value for state $s_i$ and action $a_i$.
- $Q_{\text{target}}(s_i, a_i))$ represents the target $Q$-value for state $s_i$ and action $a_i$.
- $\nabla Q_{\text{predicted}}(s_i, a_i)$ is the gradient of the predicted $Q$-value with respect to the weights.

Here is an explanation of each term and its function in Algorithm 1:

- Learning rate $\eta$: Controls the step size for updating the $Q$-network weights during training.
- Exploration parameter $\epsilon$: Balances exploration and exploitation in action selection during training.
- Discount factor $\gamma$: Determines the importance of future rewards compared to immediate rewards.
- Replay memory $D$: Stores recent transitions for experience replay to improve learning stability.
- Target network update frequency $C$: Determines how often the target $Q$-network weights are updated.
- Maximum number of episodes: Sets the limit for the number of training iterations.
- $Q$-network: Neural network approximating $Q$-values for state-action pairs.
- Bellman equation: Updates $Q$-values based on current estimates and future rewards.
- Loss function: Measures the error between predicted and target $Q$-values.
- Updated weights: New weights of the $Q$-network after each parameter update step.
- Current weights: Current weights used to estimate $Q$-values during training.
- Gradient: Direction and magnitude of steepest ascent of the loss function for weight updates.
- Online $Q$-network ($\theta_{online}$): Neural network updated during training to estimate $Q$-values.
- Target $Q$-network ($\theta_{target}$): Separate network with periodic weight updates, used for calculating target $Q$-values.
- Mini-batch: Small random sample of transitions used for training the $Q$-network in each iteration.

**Algorithm 1** Pseudocode for TLDQN Algorithm

**Inputs**: Learning rate $\eta$, Exploration parameter $\epsilon$, Discount factor $\gamma$, Replay memory $D$, Target network update frequency $C$, Maximum number of episodes
**Output**: Trained Q-network with transferred knowledge
**procedure** TLQDN(inputs)
    Initialize the Q-network with random weights: $\theta_{\text{online}}$
    **Transfer Learning:**
    - Pretrain the Q-network on the weights: $W_{\text{source}}$
    - Transfer the pretrained weights to the online Q-network: $\theta_{\text{online}} = W_{\text{source}}$

    Initialize the target Q-network with the same weights as the online Q-network: $\theta_{\text{target}} = \theta_{\text{online}}$

    **for** each episode in 1 to max_episodes **do**
        Initialize the current state: $s$
        **for** each time step in the episode **do**
            Use epsilon-greedy method to select an action: $a$
            Execute the selected action and observe the reward and next state: $r, s'$
            Store the transition $(s, a, r, s')$ in replay memory $D$
            Sample a mini-batch of transitions from replay memory $D$: *minibatch*
            **for** each transition in *minibatch* **do**
                Calculate the target Q-value using the Bellman equation (17):
                $Q_{target}(s, a) = r + \gamma \cdot \max_{a'} Q(s', a')$
                Compute the predicted Q-value using the online Q-network:
                $Q_{\text{online}}(s, a) = Q(\theta_{\text{online}}, s, a)$
                Compute the loss function using (18).
                Compute the gradient of the loss with respect to the online Q-network parameters: $\nabla L$
                Update the online Q-network weights using the learning rate $\eta$:
                $\theta_{\text{online}} = \theta_{\text{online}} - \eta \cdot \nabla L$
                Every $C$ steps, update the target Q-network weights: $\theta_{\text{target}} = \theta_{\text{online}}$
            **end for**
            Set the current state to the next state: $s = s'$
        **end for**
    **end for**
**end procedure**

### D. HYBRID INTRUSION DETECTION AND PREVENTION SYSTEM

Secure routing does not provide security to the SDWSN environment because it does not detect intrusions. Hence,

we perform hybrid intrusion detection, which includes two stages of intrusion detection, which are explained next.

### 1) MALICIOUS NODE DETECTION

The sensor nodes send their data through the controller, where the secondary controllers perform malicious or compromised node detection with the help of the investigate agent. This agent monitors and records the behaviors of the sensor nodes to increase security. The secondary controller detects the malicious nodes by collecting the investigated information and considering energy consumption, packet delivery ratio, and packet loss ratio using the **driver training-based optimization (DTO)** algorithm 2. The secondary controllers are monitored and controlled by the primary controller. The initial position of driving learners and instructors is randomly initialized, which is expressed:

$$Z = \begin{bmatrix} Z_1 \\ \vdots \\ Z_i \\ \vdots \\ Z_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} Z_{11} & \dots & Z_{1j} & \cdots & Z_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ Z_{i1} & \dots & Z_{ij} & \dots & Z_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ Z_{m1} & \dots & Z_{mj} & \dots & Z_{mn} \end{bmatrix}_{m\times n} \tag{21}$$

$$z_{i,j} = B_{Lj} + \text{ Rand. } \left(B_{Uj} - B_{Lj}\right) \tag{22}$$

where $i = 1, 2, \dots, m, j = 1, 2, \dots, n$, $Z$ represents the population DTO, $Z_i$ represents the solution of $i$-th candidate, $Z_{I,j}$ represents the $j$-th variable value which is fixed based on the solution of $i$-th candidate, $m$ represents the population size of DTO, and $n$ represents the problem variable count, and *Rand* is a random number with the range of $[0, 2]$, $B_{Uj}$ and $B_{Lj}$ are the variables representing the upper and lower bounds, respectively. The objective function of every candidate solution is calculated as:

$$O_F = \begin{bmatrix} O_{F_1} \\ \vdots \\ O_{F_i} \\ \vdots \\ O_{F_n} \end{bmatrix}_{n\times 1} \tag{23}$$

where $O_F$ represents the objective functions, and $O_{F_i}$ represents the objective function of $i$-th candidate. Candidates' solutions are obtained through three phases: training of a learner driver, learner driver patterning, and learner driver practice. The population initialization of driving instructors is defined as follows:

$$I_D = \begin{bmatrix} I_{D_1} \\ \vdots \\ I_{D_i} \\ \vdots \\ I_{D_n} \end{bmatrix}_{n\times 1} = \begin{bmatrix} I_{D_{11}} & \cdots & I_{D_{1j}} & \cdots & I_{D_{1n}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ I_{D_{i1}} & \dots & I_{D_{ij}} & \cdots & I_{D_{in}} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ I_{D_{nI_D1}} & \cdots & z_{nI_Dj} & \cdots & Z_{nI_Dn} \end{bmatrix}_{nI_D\times n} \tag{24}$$

where $I_D$ represents the driving instructor matrix, $I_{i,j}$ represents the $i$-th driving instructor of $i$-th dimension, $nI_D = [0.1n(1 - q/Y)]$ represents the driving instructor count, $q$ represents the present iteration, and $Y$ is the highest count of iterations. The current position [55] of every member is calculated as follows:

$$z_{i,j}^{P1} = \begin{cases} z_{i,j} + \text{Rand} \cdot \left(I_{D_{ki,j}} - \text{Rand}_1 \cdot z_{i,j}\right), & O_{F_{I_{Dki}}} < O_{Fi} \\ z_{i,j} + \text{ Rand } \cdot \left(z_{i,j} - I_{D_{ki,j}}\right), & \text{otherwise} \end{cases} \tag{25}$$

$$Z_i = \begin{cases} Z_i^{P1}, & O_{F_i}^{P1} < O_{F_i} \\ Z_i, & \text{otherwise} \end{cases} \tag{26}$$

where $z_{i,j}^{P1}$ represents the current position of the candidate solution in the first process of DTO, $O_{F_i}^{P1}$ represents the value of objective function, *Rand* is a random number with the interval of $[0, 1]$, $Rand_1$ is a random number with the interval of $[1, 2]$, $I_D$ is the driving instructor matrix, and $k_i$ is a number which is randomly selected from $\{1, 2, \cdots, nI_D\}$ driving instructors. In the second process, the learner driver replicates the patterns of the instructor, and the driver tries to perfect all the activities during driving using the skills of the instructor. The current position of every member with an instructor is calculated as:

$$z_{i,j}^{P2} = P \cdot Z_{i,j} + (1 - P) \cdot I_{Dki,j} \tag{27}$$

$$Z_i = \begin{cases} Z_i^{P2}, & O_{F_i}^{P2} < O_{F_i} \\ Z_i, & \text{otherwise} \end{cases} \tag{28}$$

where $z_{i,j}^{P2}$ represents the current position of the candidate solution in second process of DTO, $O_{F_i}^{P2}$ is a value of objective function, $P$ represents the index of new patterning, which is calculated as:

$$P = 0.01 + 0.9\left(1 - \frac{q}{Y}\right) \tag{29}$$

The third DTO process is based on each driver's personal training to improve its driving skills. In this stage, every member is permitted to find the best position based on a local search and its present position. The random position of the member is calculated as:

$$z_{i,j}^{P3} = z_{i,j} + (1 - 2 \text{ Rand }) \cdot \left(1 - \frac{q}{Y}\right) \cdot z_{i,j} \tag{30}$$

$$Z_i = \begin{cases} Z_i^{P3}, & O_{F_i}^{P3} < O_{F_i} \\ Z_i, & \text{otherwise} \end{cases} \tag{31}$$

where $Z_i^{P3}$ represents the current position of the candidate solution in third process of DTO, $O_{F_i}^{P3}$, is a value of objective function, and *Rand* represents the random value with the interval of $[0, 1]$. After identifying and updating the members based on the three processes, the iteration of DTO is complete. The best candidate solution is detected during implementation execution. Here, the best candidate solution is defined as the malicious node which is isolated from the network. The DTO pseudocode is given in Algorithm 2.

**Algorithm 2** Pseudocode for DTO

**Input: Investigate information**
**Output: Malicious node detection**
Correct $n$ and $Y$
Initialize the agent for interacting environment.
Initialize the population position of DTO and estimate $O_F$
**for** $q = 1$ to $Y$ **do**
  **for** $i = 1$ to $n$ **do**
    Step 1 ''Driving instructor training'': Determine $I_D$ based on the value of objective function Randomly select $I_D$ from the matrix;
    Compute the current position of DTO member using (25); Update the current position of DTO member using (26).
    Step 2 ''Learning driver patterning'': Compute the index of patterning $P$ using (29);
    Compute the current position of DTO member using (27); Update the current position of DTO member using (28).
    Step 3 ''Learning driver practice'': Compute the current position of DTO member
    using (30); Update the current position of DTO member using (31).
  **end for**
  Update the best candidate solution
**end for**

### 2) INTRUSION DETECTION

Some malicious nodes act like legitimate nodes, but they have malicious data; hence, we ensure the integrity of the message. The integrity of the data was evaluated by performing intrusion detection. Here, we detect both signature-based and anomaly-based intrusions. The secondary controller performs signature-based intrusion detection based on flow-based features (*SrcIP, DstIP, SrcPort, DstPort, etc.*) and anomaly-based intrusions are detected by evaluating packet-based features (*type of packet, type of response, connection, etc.*). In this case, **bidirectional generative adversarial network (Bi-GAN)** detects both signature- and anomaly-based intrusions and classifies the packets as normal or malicious. Fig. 6 depicts the architecture of the Bi-GAN, which is a modified version of the GAN created by adding an encoder to the traditional GAN model. The pseudo-code for Bi-GAN is given in Algorithm 3.

In a typical GAN system, the generator and discriminator neural networks compete with one another in a game-like setting, where the gains of one agent are offset by the losses of the others. Each new data sample is produced by the generative network (or generator) and then evaluated by the discriminative network (or discriminator). Another way of putting it is that the generator is trained to map random noise into a representation of the ''real data'' distribution, and the discriminator learns to tell the difference between the new datasets created by the generator (which are considered ''fake data'') and the genuine data distribution [56], [57].

The proposed Bi-GAN is able to learn the mapping concept from real data to the hidden space, thereby providing better support for the fake dataset.

In a Bi-GAN, the generator (G) and encoder (E) networks are trained simultaneously but have distinct objectives. The generator's objective is to generate realistic samples, while the encoder's objective is to map real samples to a compressed latent space representation. The discriminator (D) network's role is to distinguish between real and fake samples.

During training, the generator and encoder networks collaborate to deceive the discriminator. The generator generates fake samples from random noise, and the encoder compresses real samples into the latent space representation. The discriminator then attempts to correctly classify whether samples are real or fake.

In summary, Bi-GAN involves training the generator and encoder networks together, each with their own objectives. The generator aims to generate realistic samples, the encoder aims to map real samples to a compressed representation, and the discriminator aims to correctly classify real and fake samples. By optimizing these objectives through adversarial training, Bi-GAN facilitates the generation of realistic samples and the mapping of real samples to a compressed latent space.

Here are the mathematical expressions for the discriminator network (D), generator network (G), and encoder network (E) in more detail:

#### DISCRIMINATOR NETWORK (D)

The discriminator network takes as input a network traffic sample $x$ and outputs a probability score indicating the likelihood of the input being real or fake. It consists of multiple layers with learnable parameters (weights and biases) that transform the input and produce the final output. Let's denote the intermediate representations in the discriminator network as $h_d$, and the weights and biases as $\theta_d$ and $b_d$, respectively. The output of the discriminator network is computed as follows:

$$h_d = \text{activation}(\theta_d \cdot x + b_d)$$
$$D(x) = \sigma(W_d \cdot h_d + b'_d) \tag{32}$$

where:

- $x$ is the input network traffic sample.
- $\theta_d$ represents the weights of the first layer in the discriminator network.
- $b_d$ represents the biases of the first layer in the discriminator network.
- activation() is an activation function applied element-wise to the intermediate representation $h_d$, such as a ReLU (rectified linear unit).
- $W_d$ represents the weights of the final layer in the discriminator network.

- $b'_d$ represents the bias of the final layer in the discriminator network.
- $\sigma()$ or sigmoid() is the sigmoid activation function applied to the final layer's output to obtain the probability score.

### GENERATOR NETWORK (G)

The generator network takes as input a random noise vector z and generates a fake network traffic sample $x_{\text{fake}}$. It also consists of multiple layers with learnable parameters that transform the input noise and produce the synthetic output.

Let us denote the intermediate representations in the generator network as $h_g$, and the weights and biases as $\theta_g$ and $b_g$, respectively. The output of the generator network is computed as follows:

$$\begin{aligned} h_g &= \text{activation}(\theta_g \cdot z + b_g) \\ G(z) = x_{\text{fake}} &= W_g \cdot h_g + b'_g \end{aligned} \tag{33}$$

where:

- z is the input random noise vector.
- $\theta_g$ represents the weights of the first layer in the generator network.
- $b_g$ represents the biases of the first layer in the generator network.
- activation() is an activation function applied element-wise to the intermediate representation $h_g$, such as a ReLU.
- $W_g$ represents the weights of the final layer in the generator network.
- $b'_g$ represents the bias of the final layer in the generator network.

### ENCODER NETWORK (E)

The encoder network takes as input a network traffic sample $x$ and computes a compressed latent space representation $z_{\text{real}}$. It also consists of multiple layers with learnable parameters that transform the input and produce the compressed representation. Let us denote the intermediate representations in the encoder network as $h_e$, and the weights and biases as $\theta_e$ and $b_e$, respectively. The output of the encoder network is computed as follows:

$$\begin{aligned} h_e &= \text{activation}(\theta_e \cdot x + b_e) \\ E(x) = z_{\text{real}} &= W_e \cdot h_e + b'_e \end{aligned} \tag{34}$$

where:

- $x$ is the input network traffic sample.
- $\theta_e$ represents the weights of the first layer in the encoder network.
- $b_e$ represents the biases of the first layer in the encoder network.
- activation() is an activation function applied element-wise to the intermediate representation $h_e$, such as a ReLU.
- $W_e$ represents the weights of the final layer in the encoder network.
- $b'_e$ represents the bias of the final layer in the encoder network.

Here are the mathematical expressions that captures the training objectives in a Bi-GAN:

### DISCRIMINATOR OBJECTIVE

$$\begin{aligned} \text{loss\_D} = \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(D(x))] \\ + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))] \end{aligned} \tag{35}$$

### GENERATOR OBJECTIVE

$$\text{loss\_G} = \min_G \max_D \mathbb{E}_{z\sim p(z)}[\log(D(G(z)))] \tag{36}$$

### ENCODER OBJECTIVE

$$\begin{aligned} \text{loss\_E} = \min_E \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(D(x))] \\ + \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log(1 - D(G(E(x))))] \end{aligned} \tag{37}$$

where:

- G represents the generator network.
- D represents the discriminator network.
- E represents the encoder network.
- $p(z)$ represents the probability distribution of the random noise vector $z$.
- $p_{\text{data}}(x)$ represents the probability distribution of the real network traffic samples $x$.
- The objective is to minimize the generator and encoder objectives and maximize the discriminator objective through iterative optimization.

The generator objective aims to generate samples $G(z)$ that are classified as real by the discriminator (D). The encoder objective aims to produce compressed latent space representations $E(x)$ that can generate samples $G(E(x))$ classified as real by the discriminator (D). The discriminator objective aims to correctly classify real samples $x$ from the data distribution and fake samples $G(z)$ generated by the generator.

In the Bi-GAN algorithm, a "**concatenate**()" function is typically used in the discriminator network. The purpose of the concatenation function is to combine the features extracted from both the real samples and the reconstructed samples before passing them through the subsequent layers of the discriminator.

This function is used to merge the feature representations obtained from the encoder network (E) for the real samples and the generator network (G) for the reconstructed samples. This allows the discriminator (D) to compare and distinguish between the real and fake samples based on the combined information.

After completing the intrusion detection system, we initiate the intrusion prevention system, which helps protect the network from cyber-attacks.Â For that purpose, an optimal delegator is selected to notify the network about intrusions.

The optimal delegator is selected based on factors such as high network active time, trust, high throughput, and

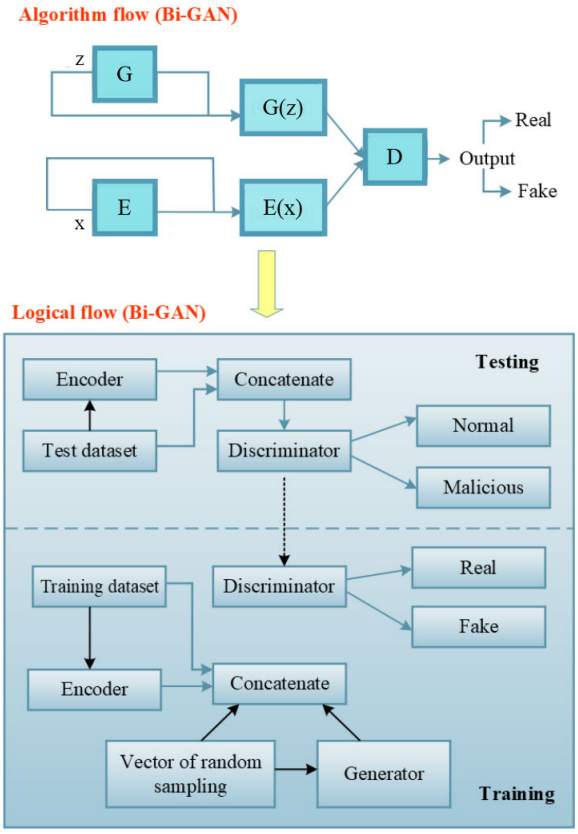



**FIGURE 6.** Architecture of Bi-GAN.

energy. This type of prevention increases the performance of intrusion detection and prevention. The Bi-GAN model is superior in terms of its ability to identify network intrusion attacks while requiring less time and effort to train. This model allows both the generator and the discriminator to be trained without the need for their training iterations to be performed in sync. By reducing the dependency between the generator and the encoder working together, we have more accurate synthetic network activity samples. This allows us to do so without having to incur excessive training overheads for the discriminator. This model is very efficient for the generator to use in order to build a synthetic network traffic sample that may make a contribution to the detection of abnormal network traffic.

### E. HIEMULTI-BLOCKCHAIN

In this research, HieMulti-Blockchain is proposed for increasing security. Here, two types of blockchains are presented, namely the main chain and the side chains, in which the main chain controls the side chains and the main chain, in this system, is used to control the entire blockchain.

A subset or partition of the overall blockchain network is known as a "shard" [58]. It is a technique used to improve scalability and enhance network performance in blockchain systems. A shard is created by dividing the blockchain network into smaller, independent units called shards. Each shard operates as a separate chain with its own set of validators and storage. Sharding allows for parallel processing of transactions, as different shards can process transactions concurrently.

**Algorithm 3** Pseudocode for Bi-GAN

**Input: Real network traffic samples** $x$
**Output: Trained Bi-GAN model: Generator network G, Encoder network E, and Discriminator network D**
Initialize the generator network G, the encoder network E, and the discriminator network D with random weights.
**Begin**
Set the learning rate $\alpha$ and the number of training iterations $N$
**for** iteration = 1 to $N$ **do**
Train the discriminator network
  **for** $k = 1$ to $K$ **do**
    Sample a batch of real network traffic samples $x$ from the data distribution
    Sample a batch of random noise vectors $z$ from the noise distribution.
    Compute the generator objective: loss_G $= -\text{mean}(\log(D(G(z))))$ by optimizing (36)
    Compute the encoder objective: x_reconstructed $= G(E(x))$
    loss_E $= -\text{mean}(\log(D(x))) - \text{mean}(\log(1 - D(\text{x\_reconstructed})))$
    Update the discriminator network:
    Concatenate the feature representations from real and reconstructed samples.
    concatenated_features = concatenate(E(x), G(z), axis=1)
    loss_D $= \text{mean}(\log(D(x))) + \text{mean}(\log(1 - D(G(z))))$ by optimizing (35)
    Update weights of $D$ using gradient descent with learning rate $\alpha$
  **end for**
  Train the generator and encoder networks
  Sample a new batch of random noise vectors $z$ from the noise distribution.
  Sample a new batch of real network traffic samples $x$ from the data distribution.
  Compute the generator objective:
  loss_G $= -\text{mean}(\log(D(G(z))))$
  Compute the encoder objective:
  x_reconstructed $= G(E(x))$
  loss_E $= -\text{mean}(\log(D(x))) - \text{mean}(\log(1 - D(\text{x\_reconstructed})))$ by optimizing (37)
  Update the generator and encoder networks:
  Update weights of G and E using gradient descent with learning rate $\alpha$
  Output the training progress
  **if** iteration % output_interval == 0 **then**
    print("Iteration:", iteration, "Generator Loss:", loss_G, "Encoder Loss:", loss_E, "Discriminator Loss:", loss_D)
  **end if**
**end for**

The purpose of introducing shards in the main blockchain is to increase the network's capacity to handle a larger number of transactions and improve throughput. By dividing the workload among multiple shards, the overall scalability of the blockchain network can be enhanced. Each shard is responsible for maintaining a subset of the blockchain's data and validating transactions related to that subset.

#### 1) MAIN CHAIN

The main chain, also known as the mainnet or parent chain, is the primary and original blockchain where the majority of transactions and blocks are recorded. It represents the core blockchain network that establishes the fundamental consensus and security mechanisms.

Miners in the main chain network validate and append new blocks to the main chain by following a consensus mechanism, typically using a process such as Proof of Work (PoW) or Proof of Stake (PoS) [59]. The process involves miners or validators performing computations (e.g., solving a cryptographic puzzle in PoW) to verify the validity of the new block. Once the validation is successful, the new block is appended to the main chain, ensuring the integrity and consistency of the blockchain network. The main chain equation is defined as:

$$MC[n] = f(MC[n-1], b[n]) \tag{38}$$

where:

- $MC[n]$ represents the updated state of the main chain after adding block $b[n]$.
- The function or algorithm $f$ is used to validate and append the new block to the main chain.
- $MC[n-1]$ represents the previous state of the main chain. It takes the previous state of the main chain ($MC[n-1]$) and the newly added block ($b[n]$) as inputs to produce the updated state of the main chain ($MC[n]$).

The main chain in a sharded blockchain typically serves as the overarching chain that coordinates and validates the state changes made in each shard. However, it does not directly merge the state changes or execute the current state of each shard. In a sharded blockchain, each shard operates independently and maintains its own state and transaction history. The main chain primarily acts as a coordinator and validator for the shards, ensuring the consistency and integrity of the overall blockchain network.

The main chain typically contains metadata, headers, or summaries of the state and transactions of each shard. It verifies and validates the transactions and state changes happening in the shards through a consensus mechanism. The main chain's role is to reach consensus on the order and validity of shard transactions and maintain the global state of the blockchain.

Here, all the transactions are stored in a hash format, which is defined as:

$$hash[n] = H(tr, Nc) \tag{39}$$

where:

- $hash[n]$ represents the hash value calculated for a specific block or data set in the blockchain.
- The hash function $H$ takes the transaction data ($tr$) and the nonce ($Nc$) as inputs and produces the resulting hash value.

#### 2) SIDE CHAINS

The proposed HieMulti-Chain includes many side chains pairs to collect all transactions into blocks and connect them together using a cryptographic hash function. This is done by making sure that the hash value of the current block is added to the next block. Because of this structure and the cryptographic hash function, the blockchain cannot be altered or reset. Every side chain includes a sequence of segments, and every segment forms a side chain block.

A conceptual grouping of blocks within the side chain is called a segment. It is typically defined based on block height, time interval, or some other criteria. The equation for a segment "$Se[n]$" is defined as:

$$Se[n] = \{b[i] : n_1 \leq i \leq n_2\} \tag{40}$$

where:

- $Se[n]$ represents the segment of the side chain, and it consists of blocks $b[i]$.
- $n_1$ and $n_2$ define the range of block indices included in the segment. The values of $n_1$ and $n_2$ will depend on the specific criteria used to define the segment, such as block height or time interval.

If we define a segment $Se[n]$ based on block height, we can have:

$$Se[n] = \{b[i] : h_1 \leq \text{block height}(b[i]) \leq h_2\} \tag{41}$$

where:

- $h_1$ and $h_2$ represent the block height range for the segment.
- $blockheight(b[i])$ refers to the height of block $b[i]$.

For a specific form of segment within a side chain,

$$Se[n] = b[n_i + 1], b[n_i + 2], \ldots, b[n_i + j] \tag{42}$$

where:

- $Se[n]$ represents the segment of the side chain, and it consists of a series of blocks starting from block $b[n_i+1]$ and continuing up to block $b[n_i + j]$.
- $n_i$ represents the index of the starting block within the side chain.
- $j$ represents the number of blocks included in the segment.

The equation selects a consecutive sequence of blocks from the side chain based on their indices.

The hash value of the side chain block is expressed as:

$$hash_{\text{side}}[n] = H_{\text{side}}(tr, Nc, prevHash[n, i], timestamp, otherParams) \tag{43}$$

where:

- The hash function $H_{\text{side}}$ takes the transaction data $tr$ and the nonce $Nc$ as inputs and produces the resulting hash value.
- $prevHash[n, i]$: The hash value of the previous block in the side chain (block $[n-1, i]$) that this block is linked to. It ensures the integrity and continuity of the chain.
- *timestamp*: The timestamp indicating the time when the block is created or added to the side chain. It helps in establishing the chronological order of the blocks.
- *otherParams*: Additional parameters may include any specific requirements or metadata associated with the side chain block, such as block height, block version, or any custom-defined parameters.

The side chain records multiple different transactions that are processed and validated independently from the main chain, allowing for increased transaction capacity and specialized functionality. The main chain in a blockchain system is typically responsible for maintaining the complete and immutable record of all transactions and data. It is designed to store the entire transaction history and provide the highest level of security and decentralization. As a result, HieMulit-Blockchian offers greater scalability and lower energy consumption than traditional blockchain.

## IV. EXPERIMENTAL RESULTS

The experimental results displayed in Fig. 1 are explained in this section. This section is divided into three subsections, namely simulation setup, comparison analysis, and research summary. The experimental results demonstrate that the proposed HieMulti-Block model achieves superior performance compared to previous works.

### A. SIMULATION SETUP

The simulation of the proposed HieMulti-Block model is performed Network Simulator 3 (NS3). The simulation of this work involves the following processes on the data plane: honeycomb-based network partitioning and clustering, RL-based intelligent routing using the TLQDN algorithm, malicious node detection using DTO, and a hybrid intrusion detection system. On the controller plane, we have HieMulti-Block, which communicates with the primary controller and later controls the secondary controllers. Bi-GAN is for classifying data packets as normal or malicious packets. On the application plane, IoT users communicate with the cloud server. Table 4 shows the system configuration, and Table 5 depicts the parameter configuration of the proposed HieMulti-Block model.

Here are explanations of how these parameters are chosen:

1) Area of Simulation (1000m × 1000m): The choice of the simulation area depends on the scale of the network intended to be modeled. A 1000m × 1000m area is suitable for modeling a moderate-sized network.
2) Modules (IPV4, Wi-Fi, Internet): The modules are chosen based on the network components we want to simulate. IPV4, Wi-Fi, and Internet modules are selected to model an IPv4-based Wi-Fi network connected to the global internet.
3) Initial Energy (100 J): Initial energy levels are set to 100 J, which is the starting energy for the nodes in our wireless sensor network. This value depends on the energy capacity of the sensor nodes we are modeling.
4) Simulation Time (600 s): Setting the simulation duration to ensure that the desired behavior and performance metrics are captured. Longer simulations can provide more statistically significant results. The simulation time is set to 600 seconds, which is the duration of the simulation.
5) Transmission Range (150 m): The transmission range represents the maximum distance over which nodes can communicate. In our case, it is set to 150 meters, reflecting the wireless range of the nodes.
6) Packet Size (64, 128, …, 1024 bytes): Multiple packet sizes are chosen to investigate how different packet sizes impact network performance. This allows us to study the effects of packet size on throughput, latency, and other metrics.
7) Channel Bandwidth (100 MHz): The channel bandwidth represents the available frequency spectrum for communication. A 100 MHz bandwidth is selected, which is a common choice for Wi-Fi networks.

**TABLE 4. System configuration.**

| Hardware Configuration | |
|---|---|
| Processor | Intel Core i5, 3GHz |
| RAM | 6 GB |
| **Software Configuration** | |
| Operating system | Ubuntu 20.04 LTS |
| Network simulator | NS-3.26 |

**TABLE 5. Network parameter configuration.**

| Parameter | Value |
|---|---|
| Number of sensor nodes | 100 |
| Number of IoT users | 10 |
| Number of edge assisted sinks | 4 |
| Number of switches | 4 |
| Number of primary controllers | 1 |
| Number of secondary controllers | 2 |
| Number of blockchain nodes | 3 |
| Number of cloud servers | 1 |
| Number of trusted authorities | 1 |
| Area of simulation | $1000m \times 1000m$ |
| Modules | IPV4, Wi-Fi, Internet |
| Initial energy | 100 J |
| Simulation time | 600 s |
| Transmission range | 150 m |
| Packet size | 64, 128, . . . , 1024 bytes |
| Channel bandwidth | 100MHz |
| Traffic type | TCP/IP, UDP |
| Packet data rate | 100Mbps |
| Packet count | 1000 |
| Transmission power | 0.005 watts |
| Mobility model | Random waypoint |

8) Traffic Type (TCP/IP, UDP): Both TCP/IP and UDP traffic types are selected. This choice allows us to study how different transport protocols affect network performance.
9) Packet Data Rate (100 Mbps): The packet data rate represents the transmission speed of data packets. A rate of 100 Mbps is chosen, which is a typical setting for high-speed communication.
10) Packet Count (1000): The packet count is set to 1000, indicating the number of packets to be transmitted during the simulation. This parameter helps us evaluate network behavior under various traffic loads.
11) Transmission Power (0.005 watts): Transmission power determines the signal strength of transmitting nodes. A value of 0.005 watts is selected, reflecting a specific power level for our nodes.
12) Mobility Model (Random Waypoint): The mobility model defines how nodes move within the simulation area. A Random Waypoint is chosen, which simulates nodes moving randomly from one point to another.

### B. IMPLEMENTATION PLAN

The network is built based on a modified honeycomb structure and consists of 100 sensor nodes, 1 trusted authority (TA), 10 IoT users, 4 edge-assisted sinks, 4 switches, 1 primary controller, 2 secondary controllers, 3 blockchain nodes, and 1 cloud node. Initially, the TA registers the sensor nodes by considering parameters like ID, PUF, location, and random information or numbers. It also registers the user by considering parameters like ID, PUF, MAC, and location. Next, the TA provides a private key with the aid of the 3D cube algorithm and performs the authentication. After completing authentication, clone attack detection is performed to increase network security.

Next, the sensor nodes are clustered based on a hexagonal structure based on a 1/7 fraction, which means each cluster includes seven hexagonal cells in the honeycomb. After completing clustering, the CH is selected based on node centrality, distance, energy, and trust. Here, we consider $w_1 + w_2 + w_3 + w_4 = 1$ and $0 \leq W_i \leq 1, \forall i, 1 \leq i \leq 4$ . In a hexagon honeycomb structure, where all the weight values are equal, it means that each criterion or parameter used for CH selection carries equal importance. In this case, the weight coefficients assigned to each criterion would be equal. Since we have four criteria ($w_1, w_2, w_3, w_4$) and all weights are equal, we assign each criterion a weight coefficient of 0.25 (1/4), we have: $w_1 = w_2 = w_3 = w_4 = 0.25$. And the sum of the four weight values should be equal to one.

Next, select the route between CH and CM for data transmission by using a TLDQN.

Then, perform hybrid intrusion detection, which consists of two stages of intrusion detection, namely malicious node detection using the DTO algorithm and intrusion detection using Bi-GAN, which classifies packets as normal or malicious as seen in Fig. 7.

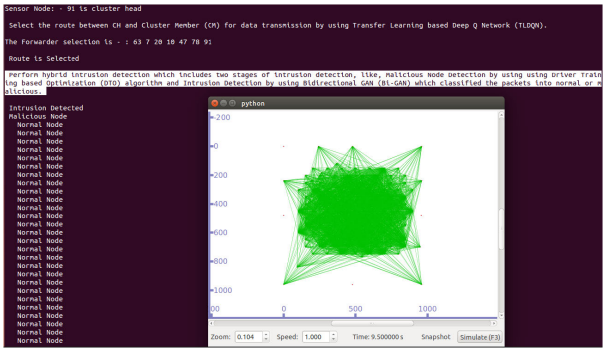



**FIGURE 7.** Normal and malicious node detection. On the left, we have the Python data visualization with information running through the network.

Select an optimal delegator to notify the network about intrusions, and open the NetAnimator to get the simulation of the system model as seen in Fig. 8.

### C. EXPLANATION OF THE CONFIGURATION SETTING FOR EACH RESULT

The performance of this research is evaluated based on the following metrics:

- Energy Consumption
- Latency
- Throughput
- Packet Delivery ratio (PDR)
- Network Lifetime
- Computation Overhead
- Detection Accuracy
- Packet Drop Ratio
- Control Overhead
- Time-Based Metrics and Confusion Matrix Analysis

#### 1) ENERGY CONSUMPTION'S CONFIGURATION SETTING

Here are detailed steps on how to measure the energy consumption in NS3:

- Select an energy model: The energy model for this simulation is `BasicEnergySource`.
- Configure energy parameters such as initial energy.
- Enable energy tracking: With NS3, allows the energy consumption is monitored and recorded during the simulation.
- Setting up intrusion detection systems, event triggers, and response mechanisms.
- Run the simulation: Execute the NS3 simulation by defining the network topology, applications, and traffic patterns.
- Collect energy consumption data.
- Analyze the results: After collecting the energy consumption data, perform analysis to calculate various metrics, such as total energy consumed by using IDS.
- Code 1 shows how the energy consumption is calculated.

Here are techniques used to identify or introduce illegitimate nodes (they are identified based on the behavior of

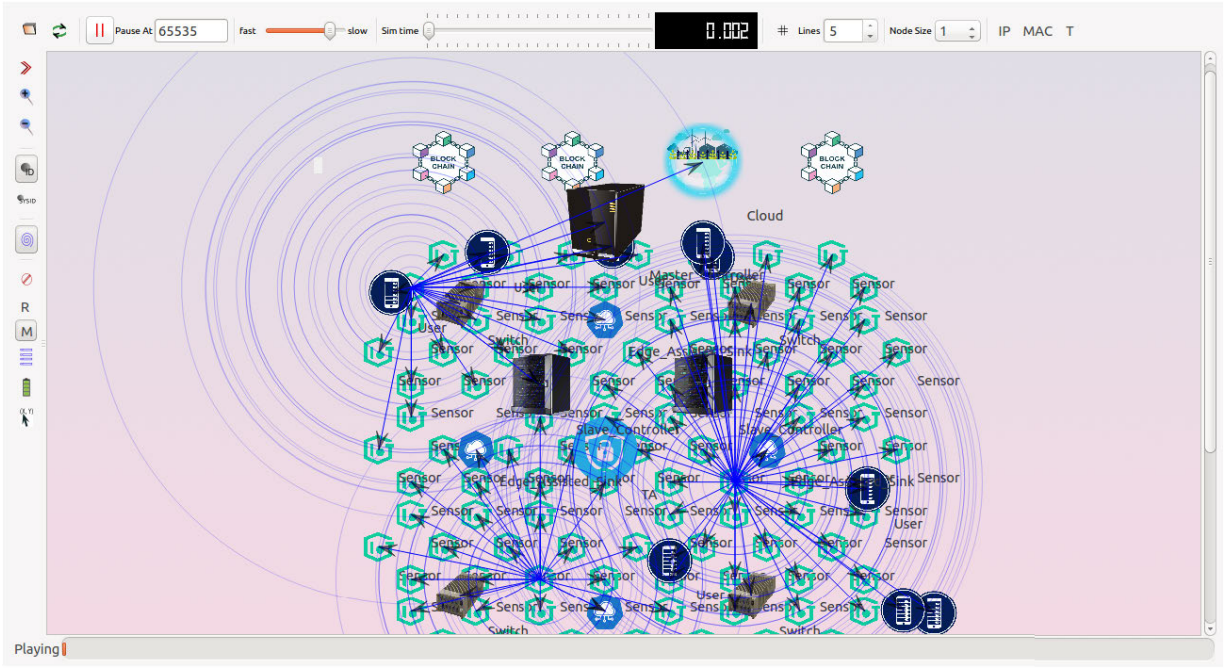



**FIGURE 8.** NS3-NetAnimator result for the network based on a modified honeycomb structure.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/energy-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("EnergyConsumption");

void MeasureEnergyConsumption(NodeContainer nodes) {
    double totalEnergyConsumed = 0.0;
    for (uint32_t i = 0; i < nodes.GetN(); i++) {
        Ptr<Node> node = nodes.Get(i);
        Ptr<BasicEnergySource> basicSource =
              node->GetDevice(0)->GetObject<BasicEnergySource>();
        double energyConsumed = basicSource->GetInitialEnergy() -
              basicSource->GetRemainingEnergy();
        totalEnergyConsumed += energyConsumed;
    }
    NS_LOG_INFO("Total Energy Consumed: " << totalEnergyConsumed << "
          Joules");
}

int main(int argc, char *argv[]) {
    NodeContainer nodes;
    nodes.Create(100);  // Create 100 nodes

    // Install energy model and set up the network topology

    // Start energy consumption measurement
    MeasureEnergyConsumption(nodes);

    return 0;
}
```

**Code 1.** Energy consumption measurement code in NS3.

the sensor nodes, which is done by the DTO algorithm that increases security):

- Node cloning: Illegitimate nodes are introduced by cloning legitimate nodes. Attackers may copy the hardware and software configuration of a legitimate node to create unauthorized replicas that appear to be legitimate. These cloned nodes can then be deployed in the network to disrupt its operations or launch attacks.
- Wireless spoofing: Attackers can impersonate legitimate nodes by spoofing their identities or emulating their communication protocols. By masquerading as legitimate nodes, the attackers can gain unauthorized access to the network, intercept communications, or inject malicious data.
- Signature-based IDS: Signature-based IDS involves comparing the behavior or characteristics of nodes with predefined signatures of known illegitimate nodes. These signatures can be generated based on prior knowledge of malicious nodes or attacks. If a node's behavior matches a known signature, it can be classified as illegitimate.
- Trust-based IDS: Trust-based detection involves evaluating the trustworthiness or reputation of nodes based on their past behavior or interactions with other nodes. Nodes with low trust levels or poor reputation scores may be considered illegitimate.
- Behavior-based IDS: Illegitimate nodes may exhibit abnormal or malicious behavior compared to legitimate nodes. By monitoring and analyzing the behavior of nodes, deviations from normal patterns can be detected. Suspicious activities or anomalies, such as abnormal power consumption, unusual data transmission patterns, or deviation from expected node behavior, can be indicators of illegitimate nodes.

### 2) LATENCY'S CONFIGURATION SETTING

Latency measurement is done through NS3. Here are detailed steps on how to measure latency in NS3:

- A network topology is set up by creating an NS3 script to define the network topology and specify the sensor nodes, their positions, and other relevant parameters.

```cpp
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/energy-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("LatencyMeasurement");

void MeasureLatency(NodeContainer nodes) {
    // Create a simple point-to-point link
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("5ms"));

    NetDeviceContainer devices = p2p.Install(nodes);

    // Install Internet stack
    InternetStackHelper internet;
    internet.Install(nodes);

    // Create a simple UDP application
    uint16_t port = 9;
    UdpServerHelper server(port);
    ApplicationContainer serverApp = server.Install(nodes.Get(1));
    serverApp.Start(Seconds(1.0));

    UdpClientHelper client(nodes.Get(1)->GetObject<Ipv4>()->GetAddress(1,
        0).GetLocal(), port);
    client.SetAttribute("MaxPackets", UintegerValue(1));
    client.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    client.SetAttribute("PacketSize", UintegerValue(1024));
    ApplicationContainer clientApp = client.Install(nodes.Get(0));
    clientApp.Start(Seconds(2.0));

    // Measure latency
    Simulator::Stop(Seconds(5.0));
    Simulator::Run();

    Time latency = Simulator::Now() - Seconds(2.0);
    NS_LOG_INFO("Latency: " << latency.GetMilliSeconds() << " ms");

    Simulator::Destroy();
}

int main(int argc, char *argv[]) {
    NodeContainer nodes;
    nodes.Create(100);  // Create 100 nodes

    // Install energy model and set up the network topology

    // Start latency measurement
    MeasureLatency(nodes);

    return 0;
}
```

**Code 2.** Latency measurement code in NS3.

- The application layer protocols, such as user datagram protocol (UDP), are defined and will generate traffic in the network.
- NS3 has the time module, which can be used to measure latency. In the NS3 script, timestamps are attached to the packets at the source node, and at the destination node, the reception time is recorded using the time module.
- The latency is calculated by subtracting the timestamp at the source node from the reception time at the destination node.
- Code 2 shows how the `LatencyMeasurement` function calculates the latency in milliseconds (ms) and prints it. The average latency is then calculated and printed at the end of the simulation.

In NS3, the average or total latency can vary for each simulation run, even with the same number of nodes. This variability is due to the inherent stochastic nature of network simulations and factors such as network conditions, traffic patterns, channel characteristics, and random events. NS3 incorporates randomness in various aspects of the simulation, such as packet transmission times, channel noise, routing decisions, and network congestion. Additionally, the simulation environment itself may introduce variability due to factors like central processing unit (CPU) load, memory allocation, or other system-level effects. As a result, running the same simulation multiple times with the same configuration can yield different latency values in each run.

```cpp
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/energy-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ThroughputMeasurement");

void MeasureThroughput(NodeContainer nodes) {
    // Create a simple point-to-point link
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));
    NetDeviceContainer devices = p2p.Install(nodes.Get(0), nodes.Get(1));

    // Install Internet stack
    InternetStackHelper internet;
    internet.Install(nodes);

    // Create OnOff applications for throughput measurement
    OnOffHelper client("ns3::UdpSocketFactory", Address());
    client.SetAttribute("OnTime",
          StringValue("ns3::ConstantRandomVariable[Constant=1]"));
    client.SetAttribute("OffTime",
          StringValue("ns3::ConstantRandomVariable[Constant=0]"));
    ApplicationContainer clientApps = client.Install(nodes.Get(0));
    clientApps.Start(Seconds(1.0));
    clientApps.Stop(Seconds(10.0));

    // Create a PacketSink application
    PacketSinkHelper sink("ns3::UdpSocketFactory", Address());
    ApplicationContainer sinkApp = sink.Install(nodes.Get(1));
    sinkApp.Start(Seconds(1.0));
    sinkApp.Stop(Seconds(10.0));

    // Measure throughput
    Simulator::Stop(Seconds(10.0));
    Simulator::Run();
    Ptr<PacketSink> sinkPtr = DynamicCast<PacketSink>(sinkApp.Get(0));
    double throughput = sinkPtr->GetTotalRx();
    NS_LOG_INFO("Throughput: " << throughput / 1e6 << " kbps");
    Simulator::Destroy();
}
int main(int argc, char *argv[]) {
    NodeContainer nodes;
    nodes.Create(100);  // Create 100 nodes
    // Install energy model and set up the network topology
    // Start throughput measurement
    MeasureThroughput(nodes);
    return 0;
}
```

**Code 3.** Throughput measurement code in NS3.

### 3) THROUGHPUT'S CONFIGURATION SETTING

Here are detailed steps on how to measure throughput in NS3:

- Data collection: Ensure that data packets are generated and transmitted from source nodes to destination nodes.
- Data reception: On the receiving node (destination), keep track of the number of data packets received within a specified time period. This can be done by counting incoming packets or using NS3's built-in counters.
- Time measurement: The simulation time is recorded upon starting and stopping measuring throughput.
- Calculate throughput: After the simulation, calculate throughput by dividing the total amount of received data (in bits) by the time taken (in seconds).
- Code 3 shows how the `ThroughputMeasurement` function calculates the total received data in bytes and prints it.

### 4) PACKET DELIVERY RATIO'S CONFIGURATION SETTING

In NS3, the PDR can be measured by utilizing the built-in tracing and logging features of the simulator.

Here are general steps on how to measure PDR in NS3 for a network with 20 to 100 nodes:

- Setting up the network by configuring the network topology and node placement in NS3, ensuring that the

```cpp
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/energy-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("PacketDeliveryRatio");

void MeasurePacketDeliveryRatio(NodeContainer nodes) {
    // Create a simple point-to-point link
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));
    NetDeviceContainer devices = p2p.Install(nodes.Get(0), nodes.Get(1));

    // Install Internet stack
    InternetStackHelper internet;
    internet.Install(nodes);

    // Create a UDP data transmission
    uint16_t udpPort = 9;
    UdpServerHelper server(udpPort);
    ApplicationContainer serverApp = server.Install(nodes.Get(1));
    serverApp.Start(Seconds(1.0));

    UdpClientHelper client(nodes.Get(1)->GetObject<Ipv4>()->GetAddress(1,
        0).GetLocal(), udpPort);
    client.SetAttribute("MaxPackets", UintegerValue(1000));  // Number of
        packets to send
    client.SetAttribute("Interval", TimeValue(Seconds(1.0)));  // Packet
        sending interval
    client.SetAttribute("PacketSize", UintegerValue(1024));  // Packet size
    ApplicationContainer clientApp = client.Install(nodes.Get(0));
    clientApp.Start(Seconds(2.0));

    // Measure packet delivery ratio
    Simulator::Stop(Seconds(10.0));  // Simulate for 10 seconds
    Simulator::Run();

    Ptr<UdpClient> udpClient = clientApp.Get(0)->GetObject<UdpClient>();
    uint32_t packetsSent = udpClient->GetPacketsSent();
    uint32_t packetsReceived = udpClient->GetPacketsReceived();
    double packetDeliveryRatio = static_cast<double>(packetsReceived) /
        packetsSent;
    NS_LOG_INFO("Packet Delivery Ratio: " << packetDeliveryRatio << "%");

    Simulator::Destroy();
}

int main(int argc, char *argv[]) {
    NodeContainer nodes;
    nodes.Create(100);  // Create 100 nodes

    // Measure packet delivery ratio
    MeasurePacketDeliveryRatio(nodes);

    return 0;
}
```

**Code 4.** Packet delivery ratio measurement code in NS3.

required number of nodes (20 to 100 nodes) are placed as required, using `NodeContainer`.

- Define the network topology.
- Set up data transmission using UDP. Create a simple sender (node 0) and receiver (node 1) to measure the packet delivery ratio.
- Measure the packet delivery ratio by comparing the number of packets sent and received.
- Run the simulation by executing it in NS3 with the specified duration and network settings.
- Calculate the PDR at the end of the simulation by retrieving the values from the packet counters and calculating the PDR by dividing the number of successfully received packets by the total number of packets sent.
- Code 4 is updated to a specific simulation scenario, including setting the `packetReceived` and `packetsSent` variables accordingly.

### 5) NETWORK LIFETIME'S CONFIGURATION SETTING

Here are general steps on how to measure network lifetime in NS3 for a network with 20 to 100 nodes:

- Network topology: Design the network layout and connectivity of nodes.

```cpp
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/energy-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("SDWSNWithIDS");

int main(int argc, char *argv[])
{
    // Enable logging
    LogComponentEnable("SDWSNWithIDS", LOG_LEVEL_INFO);
    // Create nodes
    NodeContainer nodes;
    nodes.Create(100); // Create 100 nodes
    // Install energy model
    EnergySourceContainer energySources;
    BasicEnergySourceHelper basicSourceHelper;
    basicSourceHelper.Set("BasicEnergySourceInitialEnergyJ",
        DoubleValue(10.0)); // Initial energy of 10 Joules
    energySources = basicSourceHelper.Install(nodes);
    // Set mobility model
    MobilityHelper mobility;
    mobility.SetPositionAllocator("ns3::RandomDiscPositionAllocator",
    "X", StringValue("100.0"),
    "Y", StringValue("100.0"),
    "Rho", StringValue("ns3::UniformRandomVariable[Min=0|Max=30]")); //
        Random disc placement with radius 30m
    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nodes);
    // Calculate network lifetime
    double maxRemainingEnergy = 0.0;
    for (uint32_t i = 0; i < nodes.GetN(); i++)
    {
        Ptr<Node> node = nodes.Get(i);
        Ptr<BasicEnergySource> basicSource =
            node->GetDevice(0)->GetObject<BasicEnergySource>();
        double remainingEnergy = basicSource->GetRemainingEnergy();
        if (remainingEnergy > maxRemainingEnergy)
        {
            maxRemainingEnergy = remainingEnergy;
        }
    }
    double networkLifetime = maxRemainingEnergy / (1000.0 * 0.1); //
        Assuming 0.1 W power consumption per node
    NS_LOG_INFO("Network Lifetime: " << networkLifetime << " seconds");
    return 0;
}
```

**Code 5.** Network lifetime measurement code in NS3.

- Energy model: Choose an energy model to simulate node energy consumption.
- Traffic generation: Define how nodes generate and exchange data within the network.
- Routing protocol: Select and configure a suitable routing protocol like ad-hoc on-demand distance vector (AODV) for the network.
- Performance metrics: Determine metrics to evaluate network lifetime, such as energy consumption and connectivity.
- Simulation configuration: Set simulation parameters in NS3, including the simulation duration, packet loss models, and mobility (if applicable).
- Run simulation: Execute the simulation and let it run for the specified duration.
- Data analysis: Collect and analyze simulation data to calculate network lifetime and identify failing nodes.
- Repeat and validate: Repeat the simulation with different parameters to obtain reliable results and validate findings.
- The result is stored in the `networkLifetime` variable as shown in Code 5.
- The `NS_LOG_COMPONENT_DEFINE` macro is used to print the network lifetime to the console.

### 6) COMPUTATION OVERHEAD'S CONFIGURATION SETTING

Here are general steps on how to measure computation overhead in NS3 for a network with 20 to 100 nodes:

```cpp
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/energy-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ComputationOverhead");

void MeasureComputationOverhead(NodeContainer nodes) {
    for (uint32_t i = 0; i < nodes.GetN(); i++) {
        Ptr<Node> node = nodes.Get(i);
        // Get CPU utilization information for each node
        Ptr<EnergySource> energySource =
            node->GetDevice(0)->GetObject<EnergySource>();
        double cpuUtilization = energySource->GetCpuUtilization();
        NS_LOG_INFO("Node " << i << " CPU Utilization: " << cpuUtilization);
    }
}

int main(int argc, char *argv[]) {
    // Create nodes and set up a basic network
    NodeContainer nodes;
    nodes.Create(100);  // Create 100 nodes

    // Install energy model
    EnergySourceContainer energySources;
    BasicEnergySourceHelper basicSourceHelper;
    energySources = basicSourceHelper.Install(nodes);

    // Start computation overhead measurement
    MeasureComputationOverhead(nodes);

    return 0;
}
```

**Code 6. Computation overhead measurement code in NS3.**

- Packet Processing time measurement: The timestamp is recorded when a packet is received and when the IDS finishes processing it.
- Calculation of overhead: Calculate the processing time for each packet by subtracting the timestamp when it was received from the timestamp when it was processed.
- Average computation overhead: Calculate the average computation overhead by taking the mean of the processing times for all packets.
- The `ComputationOverhead` function, in Code 6, is used to measure the computation overhead of the IDS for each packet. It calculates the time taken to process a packet by subtracting the receive time from the process time.

### 7) DETECTION ACCURACY'S CONFIGURATION SETTING

Here are steps on how to measure the detection accuracy in a network with 20 to 100 nodes:

- Determine the desired labels for events or conditions being detected, such as normal or attack scenarios.
- Run the simulation by setting up the NS3 simulation environment, including network topology, protocols, and detection mechanisms, to generate simulation results.
- Collect simulation results by gathering relevant data from the simulation run, such as packet traces or log files, to analyze the detected events or conditions.
- Analyze the results by comparing the simulated outcomes with the expected outcomes to determine the number of $T_P$, $T_N$, $F_P$, and $F_N$ cases.
- Calculate detection accuracy by using (47) to measure the detection accuracy based on the values obtained in step 4, representing a value between 0 and 1 or as a percentage.

```cpp
#include "ns3/core-module.h"

// Function to calculate detection accuracy
double CalculateDetectionAccuracy(uint32_t truePositive, uint32_t
    trueNegative, uint32_t falsePositive, uint32_t falseNegative)
{
    double accuracy = static_cast<double>(truePositive + trueNegative) /
        static_cast<double>(truePositive + trueNegative + falsePositive +
        falseNegative);
    return accuracy;
}

int main(int argc, char *argv[])
{
    // Initialize NS3
    ns3::Simulator::Run();
    ns3::Simulator::Destroy();

    // Obtain values for TP, TN, FP, FN from the simulation results
    uint32_t truePositive = 80;
    uint32_t trueNegative = 60;
    uint32_t falsePositive = 10;
    uint32_t falseNegative = 5;

    // Calculate detection accuracy
    double detectionAccuracy = CalculateDetectionAccuracy(truePositive,
        trueNegative, falsePositive, falseNegative);

    // Print the detection accuracy
    std::cout << "Detection Accuracy: " << detectionAccuracy << std::endl;

    return 0;
}
```

**Code 7. Detection accuracy measurement code in NS3.**

- The `CalculateDetectionAccuracy` function, in Code 7, takes $T_P$, $T_N$, $F_P$, and $F_N$ counts as input to calculate detection accuracy. The main function initializes NS3, retrieves the counts from simulation results, calls the `CalculateDetectionAccuracy` function, and prints the resulting detection accuracy.

### 8) PACKET DROP RATIO'S CONFIGURATION SETTING

Here are steps on how to measure the packet drop ratio in a network with 20 to 100 nodes:

- Initialize NS3 and Enable Logging. Import the necessary NS3 modules and enable logging for the `PacketSink` application.
- Create a function called `PacketDrop` to handle packet drops. This function will be called whenever a packet is dropped.
- By using `NodeContainer`, create a network with 20 to 100 nodes.
- By using `PacketSinkHelper`, set up a packet sink (receiver) and configure it to listen to a specific port.
- Install the packet sink application on the second node in the network (receiver). This node will receive and monitor incoming packets.
- Create a sender node (in this case, the first node) and a socket for sending packets.
- Set up a trace connection for the socket to the `PacketDrop` function, which will be called when a packet is dropped during transmission.
- Set up a UDP echo client on the sender node. This application will generate and send UDP packets to the receiver.
- Install the UDP echo client application on the sender node, configure its start and stop times, and specify the server IP address and port.
- Run the Simulation. Set the simulation duration and run the NS3 simulation.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/packet-sink.h"
#include "ns3/internet-module.h"
#include "ns3/wifi-module.h"

using namespace ns3;

void PacketDrop(Ptr<const Packet> p)
{
    NS_LOG_UNCOND("Packet Dropped: " << *p);
}

int main (int argc, char *argv[])
{
    LogComponentEnable("PacketSink", LOG_LEVEL_INFO);

    NodeContainer nodes;
    nodes.Create(100);

    PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), 9));
    ApplicationContainer sinkApps =
           packetSinkHelper.Install(nodes.Get(1));
    sinkApps.Start(Seconds(0.0));
    sinkApps.Stop(Seconds(10.0));

    Ptr<Node> clientNode = nodes.Get(0);
    Ptr<Socket> socket = Socket::CreateSocket(clientNode,
           UdpSocketFactory::GetTypeId());
    socket->TraceConnect("Drop", MakeCallback(&PacketDrop));

    UdpEchoClientHelper clientHelper("ServerIPAddress", 9);
    clientHelper.SetAttribute("MaxPackets", UintegerValue(1000));
    clientHelper.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    clientHelper.SetAttribute("PacketSize", UintegerValue(1024));

    ApplicationContainer clientApps = clientHelper.Install(clientNode);
    clientApps.Start(Seconds(1.0));
    clientApps.Stop(Seconds(10.0));

    Simulator::Stop(Seconds(10.0));
    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

**Code 8. Packet drop ratio measurement code in NS3.**

- Code 8 incorporates the `PacketDrop` function, which is connected to the sender's socket for handling dropped packets and logging relevant information. Throughout the simulation, occurrences of dropped packets will activate the `PacketDrop` function, thereby recording details about the dropped packets. This mechanism enables the measurement and analysis of packet drops in the network.

### 9) CONTROL OVERHEAD'S CONFIGURATION SETTING

Here are steps on how to measure the control overhead in a network with 20 to 100 nodes:

- Create a network with 100 nodes in NS-3. This can be done by creating a `NodeContainer` and using the `Create` method to generate the nodes.
- Define the network topology.
- Set up control message generation and reception. Define the message size and the interval between sending control messages.
- Create a function to send control messages periodically using the `Simulator::Schedule` method. In this case, messages are sent every second.
- Create a function to count received control messages. This function will be invoked when a control message is received by the destination node.
- Set the simulation stop time and run the simulation. This will allow control messages to be exchanged and received.
- After the simulation is complete, calculate the control overhead in messages per second. Divide the total number of received control messages by the simulation time.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/energy-module.h"
#include "ns3/internet-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ControlOverheadMeasurement");

void MeasureControlOverhead(NodeContainer nodes) {
    // Create a simple point-to-point link
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));

    NetDeviceContainer devices = p2p.Install(nodes.Get(0), nodes.Get(1));

    // Install Internet stack
    InternetStackHelper internet;
    internet.Install(nodes);

    // Create a custom control message type
    uint32_t controlMessageSize = 64; // Size of the control message in bytes
    uint32_t controlMessageInterval = 1; // Interval between sending control
           messages (in seconds)
    Ptr<Socket> source = Socket::CreateSocket(nodes.Get(0),
           TypeId::LookupByName("ns3::UdpSocketFactory"));

    // Define a function to send control messages periodically
    auto sendControlMessages = Simulator::Schedule(Seconds(1.0),
           Seconds(controlMessageInterval),
    [source, controlMessageSize](void) {
        Ptr<Packet> packet = Create<Packet>(controlMessageSize);
        source->SendTo(packet, 0,
               InetSocketAddress(nodes.Get(1)->GetObject<Ipv4>()->GetAddress(1,
               0).GetLocal(), 9));
    }
    );

    // Create a function to count received control messages
    uint32_t receivedControlMessages = 0;
    Ptr<Socket> sink = Socket::CreateSocket(nodes.Get(1),
           TypeId::LookupByName("ns3::UdpSocketFactory"));
    sink->Bind(InetSocketAddress(Ipv4Address::GetAny(), 9));
    sink->SetRecvCallback(
    MakeCallback([&receivedControlMessages](Ptr<Socket> socket) {
        receivedControlMessages++;
    })
    );

    // Measure control overhead
    Simulator::Stop(Seconds(10.0));
    Simulator::Run();

    double controlOverhead = receivedControlMessages / 10.0; // Messages per
           second
    NS_LOG_INFO("Control Overhead (Messages per Second): " <<
           controlOverhead);

    Simulator::Destroy();
}

int main(int argc, char *argv[]) {
    NodeContainer nodes;
    nodes.Create(100);  // Create 100 nodes

    // Install energy model and set up the network topology
    // Start control overhead measurement
    MeasureControlOverhead(nodes);

    return 0;
}
```

**Code 9. Control overhead measurement code in NS3.**

- Code 9 shows the `controlOverhead` variable, which represents the control overhead in messages per second, and by using `NS_LOG_COMPONENT_DEFINE`, that variable is printed.

## D. COMPARISON ANALYSIS

The performance of the HieMulti-Block model is evaluated based on various performance metrics by comparing it with its existing works namely PSO-ABC [49], OL-RL [47] and RBP-DT [48] models. The comparison is conducted by considering the energy consumption, latency, throughput, packet delivery ratio, network lifetime, computation overhead, detection accuracy, packet drop ratio, and control overhead. The results prove that the proposed HieMulti-Block model has a superior performance compared to existing works.

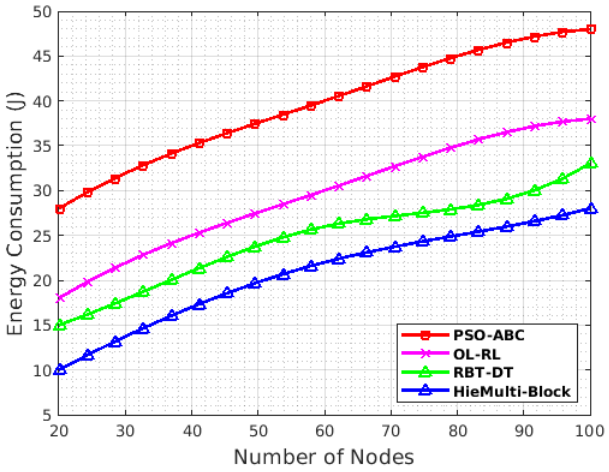



**FIGURE 9.** Comparison of energy consumption.

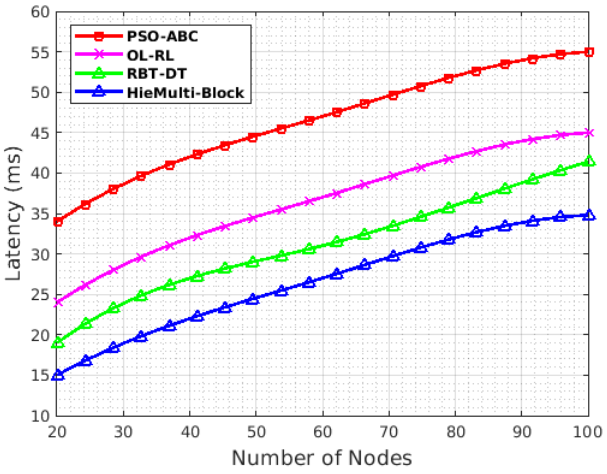



**FIGURE 10.** Comparison of latency.

### 1) IMPACT OF ENERGY CONSUMPTION

This metric is used to calculate the overall energy consumed by the system to complete the routing process, since routing is an important to reduce energy consumption. The energy consumption is calculated by subtracting the remaining energy from the total energy.

Fig. 9 represents the evaluation of energy consumption with respect to the number of nodes. The evaluation or comparison result shows that the proposed work achieves less energy consumption when compared to other existing approaches because the number of illegitimate sensor nodes was reduced, which reduces energy consumption by only considering legitimate sensors and user data rather than all the data for transmission. The number of illegitimate sensor nodes identified is 10% with respect to number of nodes. Illegitimate nodes in a WSN refer to nodes that are unauthorized or malicious, and their presence can compromise the overall security and performance of the network.

For reducing energy consumption, we performed modified honeycomb-based network partitioning and clustering, which reduces energy consumption and latency during data transmission because of its efficient network management. In addition, we used the TLDQN algorithm for routing, which reduces the learning efficiency problem by using transfer learning, which also increases training and testing speeds compared to traditional reinforcement learning. In the proposed HieMulti-Block model, we used an edge assisted sink node, whereas the edge server provides additional resources for the sink node, which reduces high energy consumption. The previous works include both legitimate and illegitimate nodes for data transmission, which increases energy consumption. In addition, the entities are placed in a random manner, which leads to poor network management and high energy consumption. The OL-RL model routed using $Q$-learning, which takes a long time to generate $Q$-values, increasing energy consumption. But in our research, the number of iterations improves the $Q$-value generation in the TLQDN algorithm.

### 2) IMPACT OF LATENCY

Latency in WSNs refers to the amount of time it takes for data or a message to transit from a source node to a destination node inside the network. The effective system must have low latency; otherwise, it does not provide efficient results. Fig. 10 represents the experimental result of latency for both proposed and existing models with respect to the number of nodes.

The comparison result proved that the proposed HieMulti-Block model achieves low latency compared to existing models. In this research, we deployed an edge-assisted sink node to reduce overloading and latency during data transmission. In addition, we proposed a high convergence algorithm for authentication, routing, and intrusion detection that reduces energy consumption due to its speed.

The existing models such as OL-RL, and PSO-ABC used slow convergence algorithms like $Q$-learning and optimization (i.e., particle swarm optimization and artificial bee colony) which increase latency and energy consumption. In our work, we construct the network based on a modified honeycomb structure, which increases network management and reduces latency during data transmission. In existing works, the network is constructed in a random manner, which increases latency due to its inefficient management.

### 3) IMPACT OF THROUGHPUT

Throughput refers to the amount of data successfully transmitted over the network within a given time period. It represents the efficiency of the network in terms of data transfer. The mathematical expression of throughput is defined as:

$$Throughput = \frac{\sum \text{Data transferred}}{\sum \text{Time taken for transmission}} \tag{44}$$

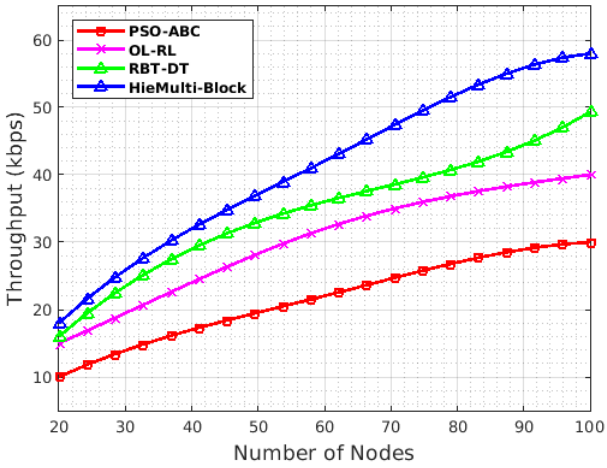



**FIGURE 11.** Comparison of throughput.

where "$\sum$ Data transferred" refers to the number of data packets that were successfully delivered to the destination node without any errors or losses during transmission. The "$\sum$ Time taken for transmission" represents the duration from the start of data transmission until the completion of the last successfully received packet. Fig. 11 shows the experimental results for throughput for both proposed models and models that already exist, based on the number of nodes. The experimental results show that the proposed HieMulti-Block achieves the highest throughput compared to existing models. In this research, we select an optimal CH based on node centrality, distance, energy, and trust that increases throughput. Furthermore, we use the TLDQN algorithm to perform RL-based intelligent routing, which first selects an optimal forwarder and then selects optimal and secure routing by taking into account various parameters such as hop count, link stability, throughput, and packet delivery ratio. This type of intelligent routing increases throughput and reduces the rate of packet loss. The proposed HieMulti-Block identifies both malicious nodes and information nodes using DTO and Bi-GAN, respectively, which increases security, therefore increasing throughput and reducing packet loss rate. The existing PSO-ABC model selects the shortest path for data transmission, which reduces the throughput because it does not consider any security measures during routing. Enhanced QoS management made possible by higher throughput ensures that crucial packets encounter less delay and are less prone to being lost. By managing congestion more effectively, and sending data more quickly, SDWSNs with higher throughput are better able to deal with the transfer of data and have lower packet loss rates. The existing OL-RL model selects the optimal path based on distance and energy, which are not sufficient for the optimal path selection that leads to less throughput. In addition, in the existing model, all the transactions are stored in a centralized manner without providing any security, which leads to poor throughput.

#### 4) IMPACT OF PACKET DELIVERY RATIO

The packet delivery ratio (PDR) is the ratio of successfully transmitted packets by the sender (source node) to the total number of packets received by the receiver (destination node). Its mathematical expression is:

$$\text{PDR} = \frac{\sum \text{Number of packet received}}{\sum \text{Number of packet sent}} \times 100\% \quad (45)$$

Note that the Bi-GAN algorithm is a deep learning model that can be trained to detect anomalous behavior or attacks in a network and potentially improve the PDR in a larger network. Bi-GAN can contribute to achieving a higher PDR in larger networks by aiding in the detection of anomalous behavior or attacks.

Here are more detailed explanations of how Bi-GAN can help in larger networks and improve the PDR:

- By utilizing Bi-GAN for anomaly detection, administrators can detect anomalies or attacks at an early stage. Early detection enables swift response and mitigation measures to be taken, reducing the impact of the anomalies on the network. By addressing anomalies promptly, the network's overall performance, including PDR, can be better preserved.
- Bi-GAN helps identify and mitigate potential attacks or malicious activities in the network. By accurately detecting and responding to such incidents, the network's security is enhanced. A more secure network reduces the likelihood of disruptions, unauthorized access, or malicious interference, which can positively impact the PDR.
- Bi-GAN detects anomalous behavior or attacks, providing insights into network abnormalities. This information optimizes resource allocation strategies by addressing issues in specific nodes or sets of nodes. Measures like adjusting transmission power, reallocating resources, or rerouting traffic can be taken to improve performance. These optimizations enhance the PDR by ensuring efficient resource utilization and avoiding areas with degraded network performance.

However, for the OL-RL model, by optimizing power allocation and data transmission decisions through reinforcement learning, the article aims to enhance the overall performance and efficiency of SDWSNs. Improving these aspects can indirectly impact the PDR by ensuring more reliable and successful packet delivery within the network. The reinforcement learning algorithms, such as $Q$-learning and deep $Q$-learning, help in making informed decisions based on environmental conditions and network requirements, which can potentially lead to an improvement in the PDR.

The PSO-ABC approach could potentially increase PDR based on the nature of these algorithms:

- Optimal cluster head selection: The improvement of CH selection can potentially lead to enhanced packet routing and reduced packet loss, thereby increasing PDR.
- Communication overhead reduction: By optimizing the routing paths and reducing unnecessary transmissions,

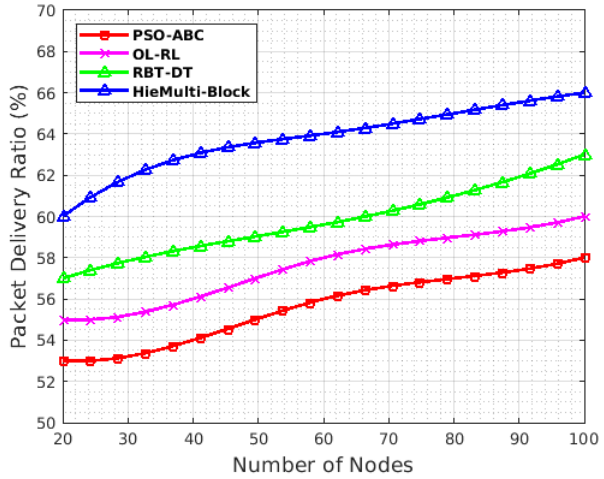



**FIGURE 12.** Comparison of packet delivery ratio.

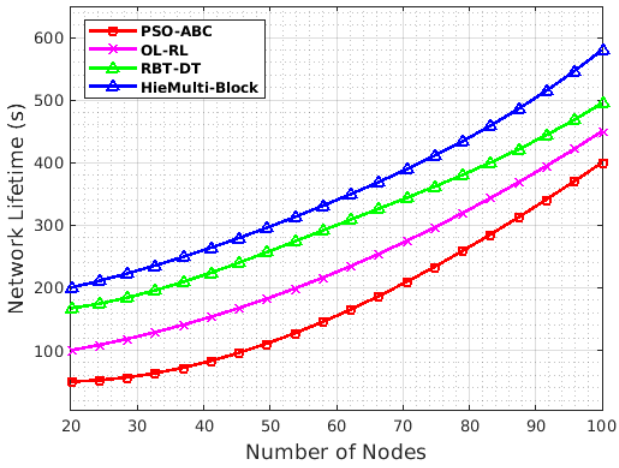



**FIGURE 13.** Comparison of network lifetime.

these algorithms can improve network efficiency. This reduction in overhead can result in more available resources for transmitting actual data packets, thereby increasing the likelihood of successful packet delivery and consequently improving PDR.

- Network performance enhancement: PSO-ABC approach improve network performance by optimizing clustering, enabling better load balancing, improved coverage, and efficient resource allocation. These enhancements create a robust network, minimizing packet loss and ensuring successful data packet delivery, ultimately positively impacting PDR.

Fig. 12 shows the packet delivery ratio with respect to the number of nodes. Based on the comparative result, it is clear that the proposed work has a higher PDR than the existing work. The proposed HieMulti-Block model performs secure authentication in an initial stage, which increases security and PDR by eliminating illegitimate entities. In addition, we perform clone attack detection by validating random numbers, which also increases the PDR. Effective network construction and management also lead to a high PDR. The optimal and secure clustering and routing process increases the PDR. In this research, with all parameters of the sensor nodes registered by the TA to ensure the legitimacy of the sensor nodes, the blockchain stores all the information with a private key provided by the TA. This process increases the security of our network and increases the PDR. In existing models, the sensor nodes are deployed in a random manner, which leads to poor network management and a lower PDR. The existing work does not perform optimal and secure clustering and routing, which increases the packet loss ratio and reduces the PDR. Lack of optimal security also reduces the PDR.

### 5) IMPACT OF NETWORK LIFETIME

This metric is utilized to evaluate the lifetime of the sensor nodes. The sensor lifetime is defined as the amount of time a sensor node remains active to survive in the environment. The longest lifetime of the sensor node increases the performance of the system. Fig. 13 depicts the comparison of network lifetime with respect to the number of nodes. The results of the figures showed that the proposed HieMulti-Block model outperformed previous works in terms of network lifetime. The proposed HieMulti-Block model achieves high network lifetime because it deploys a sensor node in a modified honeycomb-based network structure, which increases network management and reduces energy consumption; hence, it achieves high network lifetime. Here, an edge-based sink is proposed for performing clustering, routing, and investigation, which reduces overload and energy consumption in a real-time environment and increases network lifetime. We propose a high convergence algorithm, which increases processing speed and reduces waiting time and energy consumption; hence, it increases network lifetime. In existing works, sensor nodes are deployed in a random manner, which increases energy consumption and reduces network lifetime. Inaccurate clustering and CH selection also reduce network lifetime due to re-clustering and CH selection; thus, the proposed HieMulti-Block model considers multiple metrics for optimal CH selection, whereas previous works consider only a few metrics for clustering and CH selection, which reduce network lifetime.

### 6) IMPACT OF COMPUTATION OVERHEAD

The computation overhead is stated as the time taken for a sensor node to process the packets. It should be determined from the time the packet is completely received to the time it is completely processed by the sensor node. The mathematical formulation of computation overhead is defined as:

$$\text{Computation Overhead} = \frac{R_T}{C_T} \times 100\% \tag{46}$$

where $R_T$ represents the response time of packets and $C_T$ is the completion time of the sensor node. Fig. 14 depicts

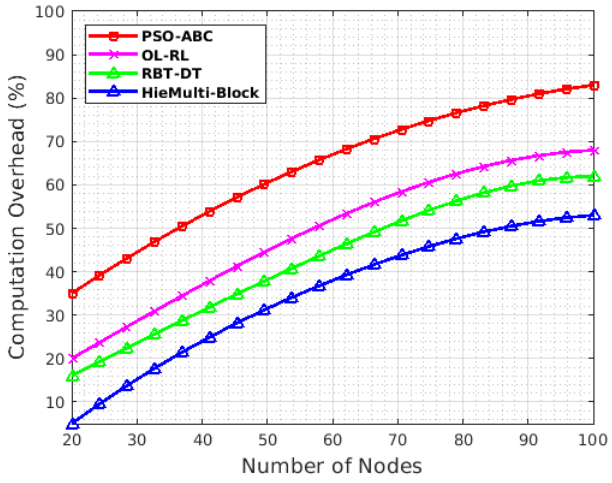



**FIGURE 14.** Comparison of computation overhead.

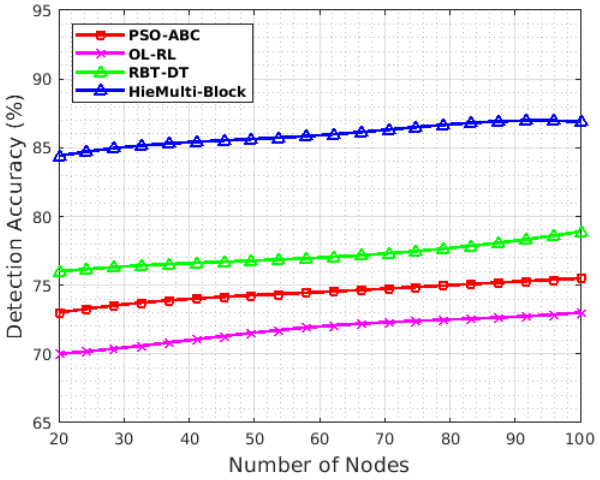



**FIGURE 15.** Comparison of detection accuracy.

comparison between proposed and existing works in terms of computation overhead for a certain number of nodes.

The result shows that the proposed work achieves lower computation overhead compared to existing models. The reason for lower computation overhead is a reduction of illegitimate nodes, efficient clustering and routing selection, and accurate detection of malicious nodes and information detection. In addition, we used a high-convergence and fast-executing algorithm, which reduces computation overhead. For instance, the 3D cube algorithm is proposed for private key generation and generates the key within a minimum amount of time using the DNN algorithm, which reduces computation overhead. Effective network management reduces the computation overhead. In this research, we proposed secondary and primary controllers for network management, which reduce the overhead in the environment. Previous works used a single controller, which does not effectively manage the environment and is unsuitable for a real-time environment due to the high overhead caused by the massive amount of task processing.

Sensor nodes carry out simple processing activities like collection, filtration, and fundamental information processing since computation overhead is negligible. The cost of computing per node is quite low. In an SDWSN, sensor nodes frequently interact with nearby nodes to transmit and combine information. Reduced processing overhead may allow for less information to be transferred between sensor nodes, lowering communication costs.

### 7) IMPACT OF DETECTION ACCURACY

This metric is applied when determining how accurate an intrusion detection system is in an SDWSN environment. The accuracy is calculated by dividing the sum of true positive and true negative by the sum of total samples. The following formula can be used to calculate the detection accuracy:

$$\text{Detection Accuracy} = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} \tag{47}$$

where $T_P$ represents true positive, $T_N$ is true negative, $F_P$ is false positive, and $F_N$ represents the false negative. Fig. 15 shows the comparison of detection accuracy with respect to the number of nodes. The comparison result demonstrates that the proposed HieMulti-Block model achieves high detection accuracy. In this research, 3D cube algorithm-based authentication is performed to increase security, which also reduces misclassification due to the presence of external attackers. Here, a malicious node is identified by using the DTO algorithm. Intrusion is identified by considering both flow-based features and packet-based features using Bi-GAN, which helps detect the intrusion accurately. For intrusion detection, existing works only consider flow-based or packet-based features, which leads to misclassification and lower detection accuracy. Furthermore, existing work does not take into account malicious nodes, which leads to lower detection accuracy because legitimate nodes may be compromised by attackers; thus, malicious node detection is also important.

### 8) IMPACT OF PACKET DROP RATIO

Packet drop ratio, also known as packet loss ratio, is a network performance metric that measures the proportion of data packets that are lost or discarded during data transfer in a computer network or communication system. It is expressed as a ratio or percentage and is an important indicator of network reliability and quality. Packet loss can occur for various reasons, including network congestion, hardware failures, or errors in the transmission process. The packet drop ratio is typically calculated as:

$$\text{Packet Drop Ratio} = \frac{\text{Number of Packets Dropped}}{\text{Total Number of Packets Sent}} \times 100\% \tag{48}$$

Each packet in a network typically has a specific deadline or an expected time frame within which it should be delivered. In cases where a packet cannot meet its deadline, network

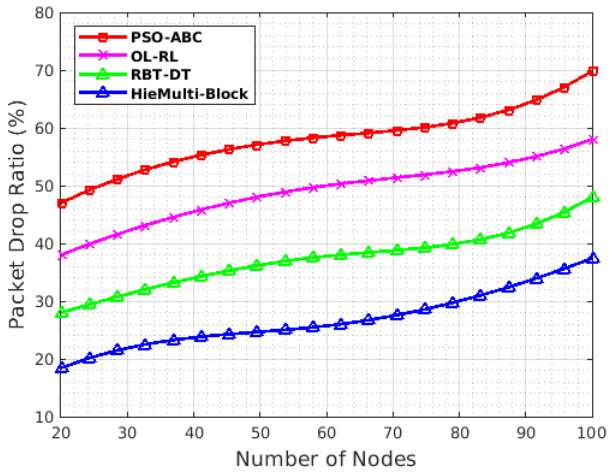



**FIGURE 16.** Comparison of packet drop ratio.

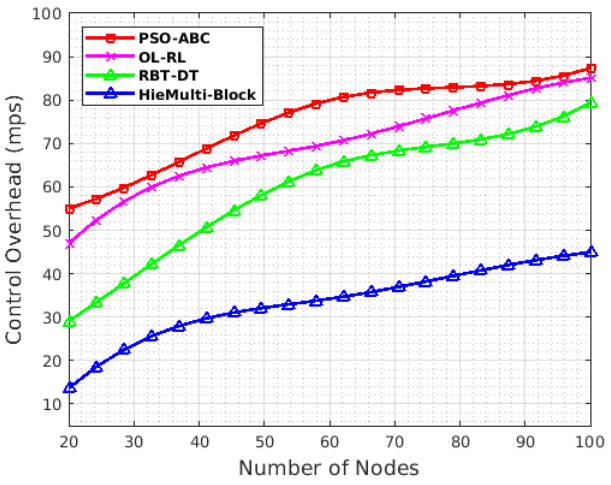



**FIGURE 17.** Comparison of control overhead.

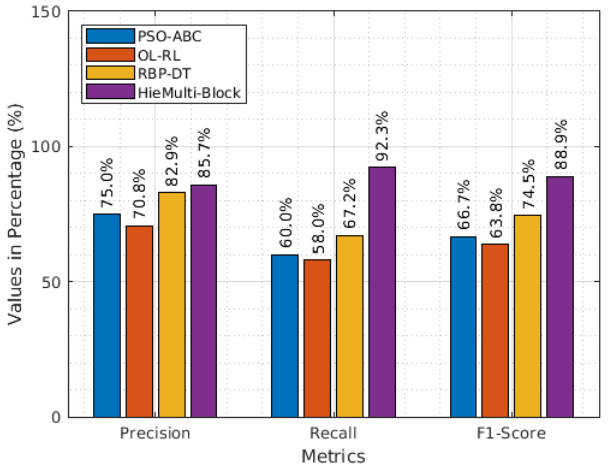



**FIGURE 18.** Time-based metrics analysis.

schedulers or protocols make efforts to minimize the number of packets that are dropped due to deadline expiration. This is crucial for maintaining network reliability and ensuring that as many packets as possible reach their destinations without being discarded.

The packet drop ratio can affect the accuracy of intrusion detection within the network. If a significant number of packets are dropped, it may result in incomplete or inconsistent data being analyzed by the IDS. This can lead to false positives or false negatives in intrusion detection, potentially reducing the security of the network.

Fig. 16 illustrates a comparison of the packet drop ratio in relation to the total number of nodes. In the proposed method, which employs reinforcement learning-based routing, efforts are made to minimize data losses during transmission. The use of RL-based routing assists in reducing the occurrence of losses by optimizing the selection of routes. By mitigating node failures and optimizing route creation, the base station is able to receive a higher volume of data packets, ultimately improving the overall network performance.

### 9) IMPACT OF CONTROL OVERHEAD

The control overhead in a network is represented by the average number of control messages sent during a specific time period, such as a network round or a given operation, and it is measured in messages per second (mps). A lower control overhead is desirable because it minimizes the energy consumption associated with sending control messages. Additionally, reducing control overhead can enhance network stability and alleviate congestion, as it reduces the non-essential data traffic in the network.

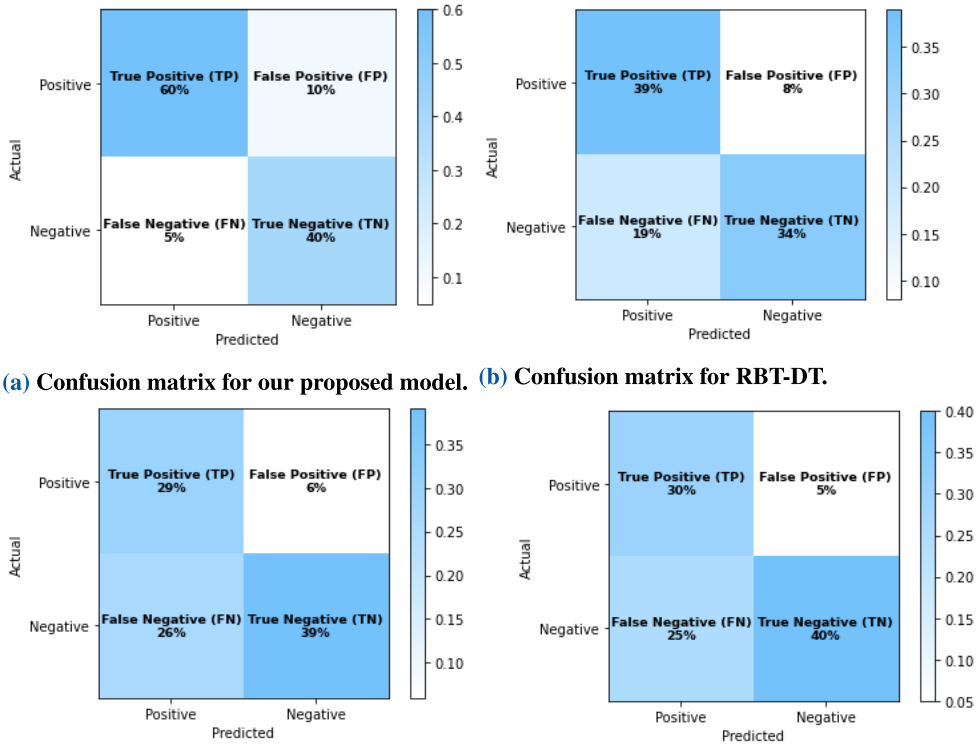
Control overhead is associated with the management, communication, and coordination required to maintain and control network operations. Each network node (such as routers, switches, or devices) generates control traffic for tasks like routing updates, address resolution, and network management.

When there are fewer nodes in a network, there are fewer devices that need to communicate with each other for control purposes. This results in reduced control overhead because there is less control traffic and fewer control messages being exchanged. In contrast, a high number of nodes, especially in a complex and large network, will generally generate more control overhead due to the increased need for coordination and communication among these nodes. This additional control overhead can affect network performance and efficiency.

Fig. 17 shows the comparison of control overhead with respect to the number of nodes. The comparison result demonstrates that the proposed HieMulti-Block model achieves lower control overhead than other models, and this results in better network performance.

### 10) MODELS' TIME-BASED METRICS AND CONFUSION MATRIX ANALYSIS

To evaluate the performance of the system, key time-based metrics such as precision, recall, and F1-score were measured and confusion matrices generated. These metrics were




(a) Confusion matrix for our proposed model.

(b) Confusion matrix for RBT-DT.

(c) Confusion matrix for OL-RL.

(d) Confusion matrix for PSO-ABC.

**FIGURE 19.** Confusion matrix visualization.

calculated throughout the 600-second simulation period to assess the system's real-time effectiveness in identifying and responding to network intrusions.

It is recommended to do the following in order to gain a deeper understanding of each model's performance: calculation of time-based metrics and confusion matrix visualization.

The precision is the ratio of true positives (correctly identified intrusions) to the total number of instances that the model predicted as positive (true positives plus false positives). It measures the accuracy of positive predictions. A high precision indicates that the model predicts positive results. The recall is the ratio of true positives (correctly identified intrusions) to the total number of actual positive instances (true positives plus false negatives). It assesses the ability to identify all relevant positive instances. A high recall indicates that the model is good at capturing positive instances. The F1-score is the harmonic mean of precision and recall, providing a balance between making accurate positive predictions (precision) and capturing all relevant positive instances (recall). It is useful for assessing the overall real-time performance of the IDS system. Table 6 shows the formulas for precision, recall, and F1-score. And Fig. 18 shows the evaluation of the key time-based metrics for all the models throughout the 600-second simulation period.

The confusion matrix [60] is a visual representation of the performance of an IDS model. It consists of a

**TABLE 6.** Formulas for Precision, Recall, and F1-Score.

| Metrics | Formulas |
|---|---|
| Precision | $\frac{T_P}{T_P+F_P}$ |
| Recall | $\frac{T_P}{T_P+F_N}$ |
| F1-Score | $2 \times \frac{Precision \times Recall}{Precision+Recall}$ |

$2 \times 2$ matrix where each cell represents one of four categories: $T_P$, $T_N$, $F_P$, and $F_N$. The confusion matrix helps in understanding the model's performance by showing how many intrusions were correctly detected ($T_P$), how many non-intrusions were correctly identified ($T_N$), how many non-intrusions were incorrectly flagged as intrusions ($F_P$), and how many intrusions were missed ($F_N$). Visualizing the confusion matrix and labeling its cells provides insights into the model's ability to make accurate predictions and distinguish between intrusions and non-intrusions. Fig. 19 shows the confusion matrix visualization for all the models.

### E. RESEARCH SUMMARY

This section depicts a summary of the proposed HieMulti-Block model's experimental results. The comparison results demonstrated that the proposed HieMulti-Block model has better performance in terms of energy consumption, latency, throughput, packet delivery ratio, network lifetime, computation overhead, detection accuracy, packet drop ratio, and

**TABLE 7. Numerical analysis of proposed and existing works.**

| Performance metrics | Proposed vs. Existing Systems | | | |
|---|---|---|---|---|
| | PSO-ABC | OL-RL | RBP-DT | HieMulti-Block |
| Energy Consumption | 47.8 J | 37.7 J | 32.8 J | 28.2 J |
| Latency | 55 ms | 44.8 ms | 41.4 ms | 34.8 ms |
| Throughput | 30 kbps | 40 kbps | 49.3 kbps | 58.2 kbps |
| Packet Delivery Ratio | 58% | 60% | 63% | 66% |
| Network Lifetime | 400 sec | 450 sec | 495 sec | 580 sec |
| Computation Overhead | 83% | 68% | 62% | 52.8% |
| Detection Accuracy | 75.5% | 73% | 78.9% | 86.9% |
| Packet Drop Ratio | 69.8% | 58% | 48% | 37.4% |
| Control Overhead | 87.34 mps | 85.08 mps | 79.32 mps | 44.96 mps |

control overhead, which are shown in Figs. 9, 10, 11, 12, 13, 14, 15, 16 and 17. The performances are achieved by performing secure authentication, honeycomb-based network partitioning and clustering, RL-based intelligent routing, and a hybrid intrusion detection system. The time-based metrics evaluation for the proposed model and other models was also conducted, as shown in 18. This evaluation also proved that HieMulti-Block has better real-time performance in comparison to other models. Table 7 depicts the average numerical values of the performance metrics for both proposed and existing approaches.

This study's most important findings are as follows:

- For increasing security, we use the 3D cube algorithm for secure authentication. This algorithm generates the private key using a DNN, which makes the system more secure and reduces energy consumption.
- For reducing energy consumption and increasing network management, we perform honeycomb-based network partitioning and clustering, which provides better coverage and throughput.
- For reducing packet loss ratio and increasing throughput, we perform RL-based intelligent routing using the TLDQN algorithm, which detects optimal and secure paths with the minimum amount of time due to applying transfer learning to the RL method.
- For enhancing security, we perform a hybrid intrusion detection and prevention system in which the malicious node is detected based on the DTO algorithm and intrusions are detected based on the Bi-GAN algorithm, which successfully detects both signature- and anomaly-based intrusions.

## V. CONCLUSION

The proposed work aims to identify the intrusions for providing security with high accuracy, throughput, and packet delivery ratio. Initially, all the sensor nodes and users are authenticated by a trusted authority to ensure their legitimacy using the 3D cube algorithm. To increase network management and reduce energy consumption, we construct the network based on a modified honeycomb structure, which also increases communication efficiency. For providing a high packet delivery ratio and throughput, we perform RL-based intelligent routing using TLDQN, which selects optimal and secure routing for data transmission and reduces the packet loss rate. Finally, a malicious node is identified by the DTO algorithm by monitoring the behavior of the sensor nodes, and both signature- and anomaly-based intrusions are detected by using the Bi-GAN algorithm, which considers both flow-based features and packet-based features for intrusion detection. The detection of both malicious nodes and intrusions leads to high security. After completing intrusion detection, prevention is initiated to increase security. Taking into account different metrics, the best delegators are chosen to let the environment know about network intrusions, which increases security. The simulation for this research is done by NS-3.26 network simulator, and the performances are evaluated based on various performance metrics, which show that the proposed work achieves better performance compared to state-of-the-art works. The proposed approach can be deployed in critical infrastructure sectors like energy, healthcare, and transportation. It can also enhance security in industrial automation systems to protect against cyber-physical attacks. The suggested system can potentially be implemented in an edge-assisted SDWSN environment; however, there are a number of problems to overcome. It needs a comprehensive approach, rigorous preparation, and significant resources. Before beginning a task like this, it will be crucial to do a feasibility assessment, handle technical issues, and take into account pragmatic concerns. Furthermore, partnering with professionals in cybersecurity, wireless networking, and artificial intelligence may greatly increase the probability of success.

## VI. FUTURE WORKS

The future work of this research project focuses on the development and implementation of two novel algorithms for IDS in IoT-enabled SDWSNs. These algorithms are expected to significantly enhance the efficiency and accuracy of intrusion detection in this context. The first algorithm will serve as a feature selection algorithm, and the second one will be used for feature extraction. These algorithms aim to address the challenges associated with processing large datasets in IoT-enabled SDWSNs and improve the overall performance of intrusion detection.

In addition to algorithm development, the research will also involve practical implementation and testing. The proposed work will include extensive experimentation using real-world data from IoT devices in SDWSNs, specifically the https://www.unb.ca/cic/datasets/nsl.htmlNSL-KDD dataset, to evaluate the effectiveness of the two novel algorithms in selecting and extracting relevant features for intrusion detection. These algorithms will be fine-tuned to ensure optimal performance and efficiency in the IoT environment, which often operates under resource constraints. Furthermore, the research will explore the integration of an advanced deep learning model as an anomaly detector in IDS, aiming to improve the accuracy of intrusion detection while conserving energy resources. By leveraging the advanced deep learning model's capacity to process and analyze

data locally on sensor nodes, this approach can enhance security without compromising efficiency. The research will investigate the trade-offs between detection accuracy and computing complexity. Ultimately, the research aims to deliver a comprehensive intrusion detection solution for IoT-enabled SDWSNs, incorporating two novel algorithms and an advanced deep learning model, with a focus on generating timely alerts to notify the users of potential security threats.

Another future work, we aim to develop an efficient communication protocol for SDWSNs that leverages advanced queuing techniques, including priority queues and weighted fair queues, to enhance QoS. Additionally, we intend to enhance security and privacy while keeping costs low by integrating 5G communication technology with attribute-based fog/edge-assisted signcryption and quantum encryption systems. Signcryption is a cryptographic technique that combines both digital signature and encryption operations into a single operation. Fog/edge-assisted signcryption is a concept that suggests the integration of signcryption techniques within fog or edge computing architectures to enhance data security and privacy in decentralized, distributed systems. This concept can involve signing and encrypting data at the edge of a network, such as on IoT devices or sensors, and may have advantages in terms of reducing the overhead associated with transmitting data to a centralized cloud server.

## APPENDIX

Our dataset associated with this paper can be accessed online at DOI: https://figshare.com/articles/dataset/SDWSNIDS_Dataset_zip/24118626

## REFERENCES


[1] E. Esenogho, K. Djouani, and A. M. Kurien, "Integrating artificial intelligence Internet of Things and 5G for next-generation smartgrid: A survey of trends challenges and prospect," *IEEE Access*, vol. 10, pp. 4794–4831, 2022.

[2] P. Aruchamy, S. Gnanaselvi, D. Sowndarya, and P. Naveenkumar, "An artificial intelligence approach for energy-aware intrusion detection and secure routing in Internet of Things-enabled wireless sensor networks," *Concurrency Comput., Pract. Exper.*, vol. 35, no. 23, Oct. 2023, Art. no. e7818.

[3] J. Kipongo, T. G. Swart, and E. Esenogho, "Design and implementation of intrusion detection systems using RPL and AOVD protocols-based wireless sensor networks," *Int. J. Electron. Telecommun.*, vol. 69, no. 2, pp. 309–318, 2023.

[4] G. A. N. Segura, S. Skaperas, A. Chorti, L. Mamatas, and C. B. Margi, "Denial of service attacks detection in software-defined wireless sensor networks," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, Jun. 2020, pp. 1–7.

[5] S. Roy, R. Dutta, N. Ghosh, and P. Ghosh, "Leveraging periodicity to improve quality of service in mobile software defined wireless sensor networks," in *Proc. IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2021, pp. 1–2.

[6] F. F. Jurado-Lasso, L. Marchegiani, J. F. Jurado, A. M. Abu-Mahfouz, and X. Fafoutis, "A survey on machine learning software-defined wireless sensor networks (ML-SDWSNs): Current status and major challenges," *IEEE Access*, vol. 10, pp. 23560–23592, 2022.

[7] A. Rahimifar, Y. S. Kavian, H. Kaabi, and M. Soroosh, "Predicting the energy consumption in software defined wireless sensor networks: A probabilistic Markov model approach," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 10, pp. 9053–9066, Oct. 2021.

[8] P. Maheshwari, A. K. Sharma, and K. Verma, "Energy efficient cluster based routing protocol for WSN using butterfly optimization algorithm and ant colony optimization," *Ad Hoc Netw.*, vol. 110, Jan. 2021, Art. no. 102317.

[9] D. L. Reddy, C. Puttamadappa, and H. N. Suresh, "Merged glowworm swarm with ant colony optimization for energy efficient clustering and routing in wireless sensor network," *Pervas. Mobile Comput.*, vol. 71, Feb. 2021, Art. no. 101338.

[10] D. Mehta and S. Saxena, "MCH-EOR: Multi-objective cluster head based energy-aware optimized routing algorithm in wireless sensor networks," *Sustain. Comput., Informat. Syst.*, vol. 28, Dec. 2020, Art. no. 100406.

[11] W.-K. Yun and S.-J. Yoo, "Q-learning-based data-aggregation-aware energy-efficient routing protocol for wireless sensor networks," *IEEE Access*, vol. 9, pp. 10737–10750, 2021.

[12] S. E. Bouzid, Y. Serrestou, K. Raoof, and M. N. Omri, "Efficient routing protocol for wireless sensor network based on reinforcement learning," in *Proc. 5th Int. Conf. Adv. Technol. Signal Image Process. (ATSIP)*, Sep. 2020, pp. 1–5.

[13] M. U. Younus, M. K. Khan, and A. R. Bhatti, "Improving the software-defined wireless sensor networks routing performance using reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3495–3508, Mar. 2022.

[14] S. Amaran and R. M. Mohan, "An optimal multilayer perceptron with dragonfly algorithm for intrusion detection in wireless sensor networks," in *Proc. 5th Int. Conf. Comput. Methodolog. Commun. (ICCMC)*, Apr. 2021, pp. 1–5.

[15] S. Amaran and R. M. Mohan, "Intrusion detection system using optimal support vector machine for wireless sensor networks," in *Proc. Int. Conf. Artif. Intell. Smart Syst. (ICAIS)*, Mar. 2021, pp. 1100–1104.

[16] R. Yao, N. Wang, Z. Liu, P. Chen, D. Ma, and X. Sheng, "Intrusion detection system in the smart distribution network: A feature engineering based AE-LightGBM approach," *Energy Rep.*, vol. 7, pp. 353–361, Nov. 2021.

[17] M. S. ElSayed, N.-A. Le-Khac, M. A. Albahar, and A. Jurcut, "A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique," *J. Netw. Comput. Appl.*, vol. 191, Oct. 2021, Art. no. 103160.

[18] M. Nkongolo, J. P. Van Deventer, S. M. Kasongo, S. R. Zahra, and J. Kipongo, "A cloud based optimization method for zero-day threats detection using genetic algorithm and ensemble learning," *Electronics*, vol. 11, no. 11, p. 1749, May 2022.

[19] A. Bhardwaj, R. Tyagi, N. Sharma, A. Khare, M. S. Punia, and V. K. Garg, "Network intrusion detection in software defined networking with self-organized constraint-based intelligent learning framework," *Meas., Sensors*, vol. 24, Dec. 2022, Art. no. 100580.

[20] A. Yazdinejadna, R. M. Parizi, A. Dehghantanha, and M. S. Khan, "A kangaroo-based intrusion detection system on software-defined networks," *Comput. Netw.*, vol. 184, Jan. 2021, Art. no. 107688.

[21] L. K. Ramasamy, K. P. F. Khan, A. L. Imoize, J. O. Ogbebor, S. Kadry, and S. Rho, "Blockchain-based wireless sensor networks for malicious node detection: A survey," *IEEE Access*, vol. 9, pp. 128765–128785, 2021.

[22] Z. Cui, F. XUE, S. Zhang, X. Cai, Y. Cao, W. Zhang, and J. Chen, "A hybrid BlockChain-based identity authentication scheme for multi-WSN," *IEEE Trans. Services Comput.*, vol. 13, no. 2, pp. 241–251, Mar. 2020.

[23] W. Meng, W. Li, and J. Zhou, "Enhancing the security of blockchain-based software defined networking through trust-based traffic fusion and filtration," *Inf. Fusion*, vol. 70, pp. 60–71, Jun. 2021.

[24] A. Rahman, M. J. Islam, S. S. Band, G. Muhammad, K. Hasan, and P. Tiwari, "Towards a blockchain-SDN-based secure architecture for cloud computing in smart industrial IoT," *Digit. Commun. Netw.*, vol. 9, no. 2, pp. 411–421, Apr. 2023.

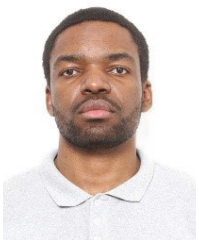
[25] J. Kipongo, E. Esenogho, and T. G. Swart, "Efficient topology discovery protocol using IT-SDN for software-defined wireless sensor network," *Bull. Electr. Eng. Informat.*, vol. 11, no. 1, 2022, Art. no. 256269.

[26] J. Kipongo, T.O. Olwal, and A. M. Abu-Mahfouz, "Topology discovery protocol for software defined wireless sensor network: Solutions and open issues," in *Proc. IEEE 27th Int. Symp. Ind. Electron. (ISIE)*, Jun. 2018, pp. 1282–1287.

[27] D. A. J. Rajan and E. R. Naganathan, "Trust based anonymous intrusion detection for cloud assisted WSN-IoT," *Global Transitions Proc.*, vol. 3, no. 1, pp. 104–108, Jun. 2022.

[28] R. A. Ramadan, "Efficient intrusion detection algorithms for smart cities-based wireless sensing technologies," *J. Sensor Actuator Netw.*, vol. 9, no. 3, p. 39, Aug. 2020.

[29] R. Goyat, G. Kumar, M. Alazab, R. Saha, R. Thomas, and M. K. Rai, "A secure localization scheme based on trust assessment for WSNs using blockchain technology," *Future Gener. Comput. Syst.*, vol. 125, pp. 221–231, Dec. 2021.
[30] R. Ramteke, S. Singh, and A. Malik, "Optimized routing technique for IoT enabled software-defined heterogeneous WSNs using genetic mutation based PSO," *Comput. Standards Interface*, vol. 79, Jan. 2022, Art. no. 103548.
[31] K. Haseeb, K. M. Almustafa, Z. Jan, T. Saba, and U. Tariq, "Secure and energy-aware heuristic routing protocol for wireless sensor network," *IEEE Access*, vol. 8, pp. 163962–163974, 2020.
[32] G. Liu, H. Zhao, F. Fan, G. Liu, Q. Xu, and S. Nazir, "An enhanced intrusion detection model based on improved kNN in WSNs," *Sensors*, vol. 22, no. 4, p. 1407, Feb. 2022.
[33] S. Jiang, J. Zhao, and X. Xu, "SLGBM: An intrusion detection mechanism for wireless sensor networks in smart environments," *IEEE Access*, vol. 8, pp. 169548–169558, 2020.
[34] L. Tan, Y. Pan, J. Wu, J. Zhou, H. Jiang, and Y. Deng, "A new framework for DDoS attack detection and defense in SDN environment," *IEEE Access*, vol. 8, pp. 161908–161919, 2020.
[35] K. Lakshmanna, N. Subramani, Y. Alotaibi, S. Alghamdi, O. I. Khalafand, and A. K. Nanda, "Improved metaheuristic-driven energy-aware cluster-based routing scheme for IoT-assisted wireless sensor networks," *Sustainability*, vol. 14, no. 13, p. 7712, Jun. 2022.
[36] S.-K. Yang, Y.-M. Shiue, Z.-Y. Su, I.-H. Liu, and C.-G. Liu, "An authentication information exchange scheme in WSN for IoT applications," *IEEE Access*, vol. 8, pp. 9728–9738, 2020.
[37] R. Bhatt, P. Maheshwary, P. Shukla, P. Shukla, M. Shrivastava, and S. Changlani, "Implementation of fruit fly optimization algorithm (FFOA) to escalate the attacking efficiency of node capture attack in wireless sensor networks (WSN)," *Comput. Commun.*, vol. 149, pp. 134–145, Jan. 2020.
[38] M. A. Khan, M. M. Nasralla, M. M. Umar, S. Khan, and N. Choudhury, "An efficient multilevel probabilistic model for abnormal traffic detection in wireless sensor networks," *Sensors*, vol. 22, no. 2, p. 410, Jan. 2022.
[39] F. F. Jurado-Lasso, K. Clarke, A. N. Cadavid, and A. Nirmalathas, "Energy-aware routing for software-defined multihop wireless sensor networks," *IEEE Sensors J.*, vol. 21, no. 8, pp. 10174–10182, Apr. 2021.
[40] G. N. Nguyen, N. H. L. Viet, A. F. S. Devaraj, R. Gobi, and K. Shankar, "Blockchain enabled energy efficient red deer algorithm based clustering protocol for pervasive wireless sensor networks," *Sustain. Comput., Informat. Syst.*, vol. 28, Dec. 2020, Art. no. 100464.
[41] T. Theodorou and L. Mamatas, "SD-MIoT: A software-defined networking solution for mobile Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4604–4617, Mar. 2021.
[42] J. Bhayo, S. A. Shah, S. Hameed, A. Ahmed, J. Nasir, and D. Draheim, "Towards a machine learning-based framework for DDOS attack detection in software-defined IoT (SD-IoT) networks," *Eng. Appl. Artif. Intell.*, vol. 123, Aug. 2023, Art. no. 106432.
[43] S. Siddiqui, S. Hameed, S. A. Shah, I. Ahmad, A. Aneiba, D. Draheim, and S. Dustdar, "Toward software-defined networking-based IoT frameworks: A systematic literature review, taxonomy, open challenges and prospects," *IEEE Access*, vol. 10, pp. 70850–70901, 2022.
[44] M. Khalid, S. Hameed, A. Qadir, S. A. Shah, and D. Draheim, "Towards SDN-based smart contract solution for IoT access control," *Comput. Commun.*, vol. 198, pp. 1–31, Jan. 2023.
[45] M. Alotaibi, "Improved blowfish algorithm-based secure routing technique in IoT-based WSN," *IEEE Access*, vol. 9, pp. 159187–159197, 2021.
[46] I. A. A. E. And and S. M. Darwish, "Towards designing a trusted routing scheme in wireless sensor networks: A new deep blockchain approach," *IEEE Access*, vol. 9, pp. 103822–103834, 2021.
[47] M. U. Younus, M. K. Khan, M. R. Anjum, S. Afridi, Z. A. Arain, and A. A. Jamali, "Optimizing the lifetime of software defined wireless sensor network via reinforcement learning," *IEEE Access*, vol. 9, pp. 259–272, 2021.
[48] P. Srividya and L. N. Devi, "An optimal cluster & trusted path for routing formation and classification of intrusion using the machine learning classification approach in WSN," *Global Transitions Proc.*, vol. 3, no. 1, pp. 317–325, Jun. 2022.
[49] L. Sixu, W. Muqing, and Z. Min, "Particle swarm optimization and artificial bee colony algorithm for clustering and mobile based software-defined wireless sensor networks," *Wireless Netw.*, vol. 28, no. 4, pp. 1671–1688, May 2022.
[50] M. Huang, B. Yu, and S. Li, "PUF-assisted group key distribution scheme for software-defined wireless sensor networks," *IEEE Commun. Lett.*, vol. 22, no. 2, pp. 404–407, Feb. 2018.
[51] J. Jin and K. Kim, "3D CUBE algorithm for the key generation method: Applying deep neural network learning-based," *IEEE Access*, vol. 8, pp. 33689–33702, 2020.
[52] S. Rameshkumar, R. Ganesan, and A. Merline, "Progressive transfer learning-based deep q network for DDOS defence in WSN," *Comput. Syst. Sci. Eng.*, vol. 44, no. 3, pp. 2379–2394, 2023.
[53] H. Alavizadeh, H. Alavizadeh, and J. Jang-Jaccard, "Deep Q-learning based reinforcement learning approach for network intrusion detection," *Computers*, vol. 11, no. 3, p. 41, Mar. 2022.
[54] M. Kaya and R. Alhajj, "Utilizing fuzzy OLAP mining towards novel approach to multiagent modular reinforcement learning," in *Proc. IEEE/WIC/ACM Int. Conf. Intell. Agent Technol. (IAT)*, Sep. 2004, pp. 197–203.
[55] R. S. Arunkumar and A. Chinnasamy, "NLOS node localization for improving warning message distribution in VANETS," *Concurrency Comput., Pract. Exper.*, vol. 35, no. 23, Oct. 2023, Art. no. e7816.
[56] W. Xu, J. Jang-Jaccard, T. Liu, and F. Sabrina, "Training a bidirectional GAN-based one-class classifier for network intrusion detection," 2022, *arXiv:2202.01332*.
[57] W. Xu, J. Jang-Jaccard, T. Liu, F. Sabrina, and J. Kwak, "Improved bidirectional GAN-based approach for network intrusion detection using one-class classifier," *Computers*, vol. 11, no. 6, p. 85, May 2022.
[58] N.-Y. Lee, "Hierarchical multi-blockchain system for parallel computation in cryptocurrency transfers and smart contracts," *Appl. Sci.*, vol. 11, no. 21, p. 10173, Oct. 2021.
[59] Y. Chen, X. Yang, T. Li, Y. Ren, and Y. Long, "A blockchain-empowered authentication scheme for worm detection in wireless sensor network," *Digit. Commun. Netw.*, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864822000566, doi: 10.1016/j.dcan.2022.04.007.
[60] O. Caelen, "A Bayesian interpretation of the confusion matrix," *Ann. Math. Artif. Intell.*, vol. 81, nos. 3–4, pp. 429–450, Dec. 2017.



**JOSEPH KIPONGO** received the B.Tech. degree in electrical engineering (light current) from the Tshwane University of Technology, South Africa, in 2017, and the M.Phil. degree in electrical and electronic engineering science from the University of Johannesburg, South Africa, in 2022, where he is currently pursuing the Ph.D. degree in electrical and electronic engineering science.

He has been lecturing and tutoring undergraduate students for more than five years in engineering with the University of South Africa, and has worked for various telecommunications and electronics companies. His research interests include telecommunications, artificial intelligence, electronics, data science, cybersecurity, digital signal processing, digital communications, the Internet of Things, wireless sensor networks, 5G wireless networks, software-defined networks, machine learning, and control systems.

**THEO G. SWART** (Senior Member, IEEE) was born in South Africa. He received the B.-Ing. and M.-Ing. degrees (cum laude) in electric and electronic engineering from Rand Afrikaans University, South Africa, in 1999 and 2001, respectively, and the D.-Ing. degree from the University of Johannesburg (UJ), South Africa, in 2006.

He is an Associate Professor with the Department of Electrical and Electronic Engineering Science and the Director of the UJ Center for Telecommunications. To date, he has over 20 journal articles and over 50 conference papers to his name and has been the co-editor of and contributor to two editions of a comprehensive book on power-line communication. His research interests include information theory, error correction coding, line coding, digital communications, and power-line communications.

Prof. Swart was the Chair of the IEEE South Africa Chapter on Information Theory. He has served as the Co-Chair for the African Winter School on Information Theory and Communications, from 2015 to 2019; the Technical Program Co-Chair for the IEEE International Symposium on Power Line Communications and its Applications, in 2013; and the Technical Program Track Co-Chair for IEEE Africon, in 2017. He has been a TPC member of various other international conferences. He has done reviews for various journals and conferences, including reviews for IEEE TRANSACTIONS ON INFORMATION THEORY, IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE International Symposium on Information Theory, IEEE International Symposium on Power Line Communications and its Applications, IEEE Africon, and SATNAC. He is a Specialist Editor of *Communications and Signal Processing for the SAIEE Africa Research Journal*.

**EBENEZER ESENOGHO** (Member, IEEE) received the Diploma degree (Upper-Div.) in computer engineering, the B.Eng. degree (Hons.) (Upper-Div.) in computer engineering, and the M.Eng. degree (Upper-Div.) in electronics/telecommunication engineering from the University of Benin, in 2003, 2008, and 2012, respectively.

He lectured with the University of Benin. Previously, he has involved in research and teaching with the Centre for Radio Access and Rural Technology, University of KwaZulu-Natal, and the Centre of Excellence, where he rounded up the Ph.D. degree in electronic (5G cognitive network) from 2013 to 2016, funded by Telkom SA, Eskom, Alcatel, Ericsson, and Huawei. He was hired into the prestigious Global Excellence and Stature (GES) Post-Doctoral Research Fellowship with the University of Johannesburg, Auckland Park, under the Institute for Intelligent System (IIS), Center for Telecommunication Research (CfT) to pursue further research. Currently, he doubles as a Senior Research Associate with CfT and a Research Project Coordinator of Liquid Telecom Research Hub. He has authored/coauthored several peer-reviewed journals and conference papers. His research interests include fifth generation (5G) wireless networks, cognitive radio networks, network security/cybersecurity, smart grid networks, the IoT/IoE, SDN/SDR, wireless sensor networks, artificial intelligence/machine learning, big data and mobile/cloud computing, and visible light communication.

Dr. Esenogho is a member of the South Africa Institute of Electrical Engineers (SAIEE) and the IEEE Region 8. He was a recipient of several grants/scholarships/awards/fellowships, including the CEPS/Eskom's HVDC 2013; the CEPS/Eskom's HVDC 2014; the J. W. Nelson 2015; and GES 2017, 2018, and 2020. He won the Best Oral Paper Presentation at the Postdoctoral Conference, during a postdoctoral fellowship, in 2019. He is passionate about service to humanity and has engaged in several community services both in South Africa and his home country. His leadership service is evident as the First Postgraduate Representative for the Student Engineering Council in the Faculty Management Board of the University of KwaZulu-Natal, during the Ph.D. study, from 2016 to 2017. He also served as the first black Research Representative in the University of Johannesburg Senate/Council, from 2018 to 2021. He was a UJ/DST/NRF Research Delegate to the H2020-ESASTAP EU-South Africa STI Cooperation on Strengthing Technology, Research, and Innovation in Vienna, Austria, to woo investors. He has chaired sessions at conferences and reviewed for some notable ISI/Scopus journals in his field. He is very familiar with ISO-9001/18001/14001/22301/27001/20000-1 with a strong affinity for projects linked to research and development. He is a registered Engineer in Nigeria.