LEX LOCALIS-JOURNAL OF LOCAL SELF-GOVERNMENT
ISSN:1581-5374E-ISSN:1855-363X
VOL.23,NO.S5(2025)

REAL-TIME TRAFFIC ACQUISITION AND FLOW-LEVEL FEATURE EXTRACTION USING A REALTIME-NET FLOW EXTRACTOR (RTNFE)

Dr.K.K.Savitha[1], T.Raja[2], H.Fathima[3]

[1]Assistant Professor, Department of Computer Applications,
Bharathiar University PG Extension and Research Centre, Erode.
[2]Research Scholar, Department of Computer Applications,
Bharathiar University PG Extension and Research Centre, Erode.
Research Scholar, Department of Computer Applications,
Bharathiar University PG Extension and Research Centre, Erode.

savitha.pge@buc.edu.in[1],
savitha.gopinath@gmail.com[1]
vtrajan3@gmail.com[2]
fathi.fathimahussain@gmail.com[3]
*Corresponding Author mail id: vtrajan3@gmail.com

Abstract: As cyber threats become more advanced, constantly monitoring network traffic is important for detecting intrusions and stopping them. A new RealTime-NetFlowExtractor (RTNFE) framework was created using Python and combines Scapy, Kafka, and Wireshark through PyShark to read packets in real-time and organize them by flow levels. Because RTNFE has a live-streaming feature and instant buffering, it offers real-time analytics of packets. The features like timestamp, IP addresses of each end, ports, protocol, and counts of bytes and packets, along with flow duration, are all extracted using a parallel, sized sliding window. To simulate real attacks, CICIDS 2018 packets are played back using tcpreplay, which contains both normal and malicious retrieved traffic that is further classified using mathematically modified deep learning technique. Throughput measures the number of packets per second, time to analyze each feature indicates latency, and data concerning packet-to-flow completeness is used for evaluation. According to the outcome, such a system is a good way to perform real-time analytics and can be used in downstream functions such as finding unusual patterns in networks or stopping new attacks.

Keywords: Packet capture, flow extraction, sliding window, kafka streaming, replay evaluation, feature normalization, attention mechanism, deep learning classification, and adam optimization.

Introduction

Real-time traffic acquisition captures and processes live network data to extract flow-level features, enabling immediate detection of anomalies and cyber threats. This continuous monitoring strengthens network security by providing timely insights, facilitating proactive intrusion detection, and ensuring resilient defense against evolving attacks in dynamic network environments. As more services go digital and many devices are connected, the vast growth of internet traffic is making serious pressure on today's communication networks to work well and be secure. The threats have developed very quicklyacross the connected network [1]. Cybercriminals now use Advanced Persistent Threats (APTs), ever-changing malware, and recently discovered security weaknesses to break into networks and steal important data [2]. The improved techniques of attackers have made it harder for the traditional intrusion detection approach to be successful. Because of this challenge, those managing networks and analysing cybersecurity are resorting to continuous examination of network activity and in-depth data analysis, which supports faster action [3, 4].

Flow analysis studies statistics and time periods in communication among different network devices, so it is more precise than analysing packets at the lowest level [5]. This part of the flow includes details such as IP addresses, ports, protocols, timestamps, the number of bytes exchanged, the number of packets transferred, and the session's duration. Those like Cisco's NetFlow and the IETF's IPFIX are some of the recognized structures used to display this data. But these applications usually include special software that cannot be easily customized

or changed. Besides, common systems usually run offline, process multiprocessor analyses from stored records, and analyze them in batches, which makes them unfit for reacting to modern emerging threats [6].

The security community is using open-source tools such as Scapy, Wireshark and its Python version (PyShark), and Apache Kafka. With Scapy, it is possible to change packet contents flexibly and use many protocols, and PyShark offers Wireshark's powerful features for detailed viewing. The queue that Kafka offers can handle data packets without involving the processing step immediately. Even though these tools are strong individually, it is still tough to assemble them into one real-time pipeline. Problems such as dropped packets, off-sync threads, and the great amount of time taken for deep parsing may decrease both the system's efficiency and scalability [7, 8].

This research intends to form a single, low-latency, and modular framework in Python to address the issues mentioned and make real-time feature extraction for machine learning-based intrusion detection. At present, leading researchers use CICIDS 2017, NSL-KDD, and UNSW-NB15 among other static datasets that are dated and not diverse enough for effective model training in real-life settings. Then again, using a real-time system means you can capture or replay traffic in real time to label it while Ethernet is still up and running. Besides, real-time analysis means that protection measures can be put in place sooner after detection.

The intended result is RTNFE, a Python module that gathers traffic in real time or through a replay, analyzes the packets, and turns them into data at the flow level. Scapy is used for packet sniffing, PyShark helps to parse packets in detail, and Kafka is the solution for efficient queuing of data. The engine uses both asyncio and Python threads to do all the capture, buffering, and flow aggregation tasks. Building accurate flow records is made possible by the main part of the flow engine, a sliding time-window model. The features included in this process are duration of flow, number of packets or bytes per flow, the time between consecutive packets, and the direction of each flow's packets [9].

To make our behavior similar to real-life, we rely on the CICIDS 2018 dataset that provides numerous labeled PCAPs with attacks like DoS, DDoS, Botnet, Brute-force, and Web attacks. With tcpreplay, the process of replaying them is speed-limited, so the system acts as if it is processing them from a live connection. There is a topic between the capture and analysis modules in Kafka that makes the process more independent and tolerant to failures.

The purpose of this research is to deal with an important issue in the current cybersecurity monitoring system. With its open-source tools, high speed, use of multiple threads, and ability to capture, analyze, and feature real-time data, RTNFE makes a notable improvement to adaptable network defense systems. Adopting this method makes it possible to do more advanced studies on real-time problem detection, forecasting, and automatic responses to problems based on using data efficiently.

Literature Review

The table 1 shows what recent studies (2024–2025) have researched about IDS that rely on advanced machine learning techniques and optimization on CICIDS benchmark datasets. Such works concentrate on better detection accuracy, choosing the right features, and helping analyze packets almost instantly. Most research studies use AI methods like bio-inspired and deep learning, yet they often handle the data outside of real-life conditions. The summary provided in the table allows for spotting similarities in methodology, slow areas, and problems universities have, this foundation was key in building RTNFE.

Table 1. Comprehensive Analysis of Literature Review

Author(s) & Year	Algorithm / Technique Used	Performance Metrics	Core Contribution / Findings
Najafi Mohsenabad & Tut (2024) [10]	Bio-Inspired Feature Selection: ACO, ABC, FPA + ML Classifiers	ACO: 99.0% Accuracy FPA: 98.7% Accuracy, 1s build time ABC: 98.6% Accuracy	Proposed optimized IDS with minimum features; ACO proved most effective; reduced model building time with FPA.
Selvam&Velliangiri (2024) [11]	CNN + Deep Autoencoder (AE) + RF	Precision: 99.5% Recall: 99.5% F1-score: 99.5%	Demonstrated superior deep learning-based intrusion detection on CICIDS2017/2018; efficient timestamp-based filtering.
Gopalsamy (2024) [12]	MLP-Backpropagation (MLP-BP)	Accuracy: 98.97% Precision: 99% Recall: 98% F1-score: 99.38%	Evaluated MLP-BP on normalized and cleaned CICIDS2018; showed balance in detection metrics.
Ibrahimi et al. (2024) [13]	Decision Tree, Random Forest, Naïve Bayes, Gradient Boost	Accuracy (varied by model): 96%–99%	Compared supervised ML classifiers for both binary and multiclass IDS tasks; suitable for IoT threat adaptation.
Chimphlee&Chimphlee (2024) [14]	Hyperparameter-Optimized XGBoost (HO-XGB)	Accuracy: >99% (with tuned params)	Demonstrated that HO-XGB outperforms conventional methods by fine-tuning 10+ hyperparameters.

Onishchenko et al. (2024) [15]	Associative Rule Mining + Intelligent Data Analysis (IDA)	Qualitative Evaluation; no exact metric	Proposed automated rule generation for cyber incident detection in SIEM using CICIDS2018; tackled dimensionality and rule sequencing.
Kumar & Pandey (2024) [16]	Comparative Survey: ANN, RF, CNN, Apache Spark	RF: 99.98% Accuracy ANN: 99.97% Accuracy	Provided empirical performance comparison showing CICIDS2018 superiority over CICIDS2017 in IDS development.
Khan et al. (2024) [17]	Literature Study on Anomaly Detection Using ML	Not applicable	Highlighted the insufficiency of CICIDS2017 and recommended dynamic datasets like CICIDS2018 for ML generalization.
Dube (2024) [18]	Dataset Integrity Evaluation	Not applicable	Criticized use of summarized CICIDS2017 data; advocated for using raw CICIDS2018 to avoid model overfitting.

The main focus of most current intrusion detection frameworks is to reach a high rate of accuracy by working with static data and doing batch processing. Although Artificial Bee Colony (ABC) [19], Ant Colony Optimization (ACO), Deep Autoencoders (AE), CNNs, and XGBoost have given promising outcomes, they are mostly judged with offline methods. The result is that they cannot be applied effectively in fast and urgent networks. Besides, most of the time, these approaches use features extracted in advance, so they cannot handle threats that change swiftly.

High efficiency is lacking when it comes to real-time packet analysis, dynamic collection of data flows, and fast processing of feature data at the same time. There is not much attention paid to making architectures that can be easily used with streaming technologies such as Kafka. A lot of designs ignore factors such as latency and throughput, choosing to measure security solutions only by their accuracy—but accuracy is not important in actual SOC operations or IoT-based infrastructure.

RTNFE fixes these limitations with the combination of Scapy for capture, PyShark for parsing, and Kafka for handling asynchronous data streams. With parallel sliding-windows,

data is both grouped and extraction occurs in real time, changing the raw traffic into organized formats similar to NetFlow. PCAP replay on RTNFE works with CICIDS 2018 data, which allows investigators to move between controlled testing and using the system in real life. As a result, IDS research changes from looking at events that have already occurred to spotting them as they happen.

Proposed Methodology

To use the proposed RTNFE, line capture of packets, their processing, and detection of intrusions is being done by using deep learning. Via PyShark, all the packets are stored and the Scapy, then investigates them to identify details including timestamp, IPs, ports, protocol, number of packets or bytes, and the time taken during the flow. A rolling window of data is used so that these features can be calculated in real time with no errors. At the second stage, data is sent over Kafka from the first stage so that the second-stage system can take it in for analysis as required. At this point, tcpreplay is used to run the CICIDS 2018 traffic and assess the framework's behavior with both normal and malignant traffic, the latency it incurs, supported data rate, and packet delivery effectiveness. The next stage of the process includes a CNN with BiLSTM layer and normalizes the flow vectors using a better mathematical approach. CNN deals with immediate surrounding places, while BiLSTM processes relationships that exist over longer time periods. At last, a Softmax classifier helps determine if a flow is dangerous or not. Balance weighting for each class is applied to the loss function to tackle problems of class imbalance. With this setup, discovering cyber-attacks happens promptly, effectively, and it is flexible to use in security systems nowadays.

Packet Capture & Flow-Level Feature Extraction

Packet Capture & Flow-Level Feature Extraction is the main step in the RTNFE framework that gathers raw network packets and changes them into flow-level data in real time. First, PyShark is used to access TShark, so one does not have to parse captures manually when doing packet analysis with Python. At the same time, Scapy is utilized to parse every packet, acquiring useful information from its Ethernet, IP, TCP/UDP, and application header fields. All these packets have the timestamp, the IP addresses, the port numbers, the protocol, the size, and direction of the packet extracted during the parsing.

Packets are grouped in a set interval by a sliding window technique, and the grouping is based on the combination of their 5-tuple (source and destination IP addresses, source and destination ports, and protocol) and how far apart they are in the packet sequence. Flow_Duration, Total_Bytes, Packet_Count, Avg_Packet_Size, and Inter_Packet_Arrival_Time are the metrics that describe every flow. These flow records are saved in buffers and they are timed out so that buffer overflow does not occur.

Also, the logic for extracting information is designed so that packets from each interface may be caught by separate threads, which maximizes throughput and makes sure only a small number of packets are lost at high speeds. It is necessary to perform packet-to-flow conversion to provide high-speed analytics and make sure that fast detection works in environments with high packet volumes. These feature vectors are used to show the results live and classify intrusions with a deep learning analysis after extraction.

Kafka-Based Streaming & Buffering

Kafka-Based Streaming & Buffering in the RealTime-NetFlowExtractor (RTNFE) framework enables low-latency, fault-tolerant transmission of flow-level features from the capture module to downstream components such as classification and storage. Apache Kafka is used as a distributed publish-subscribe messaging system, facilitating scalable data streaming in real time. Each processed flow Fi is structured as a JSON object containing:

Fi
={src_ip,dst_ip,src_port,dst_port,protocol,flow_duration,pkt_count,byte_count,avg_pkt_size,
iat_mean}

This tuple is serialized and sent by a Kafka producer to a designated topic Tc, representing class c∈{benign,malicious}.
Let the message production rate is given in Equation 1.

$$\lambda_p = \frac{N_f}{\Delta t}$$

where N_f is the number of flows generated in time window Δt. This must satisfy the Kafka broker throughput capacity λ_b to avoid queuing delay is given in Equation 2.

$$\lambda_p \leq \lambda_b$$

Kafka brokers maintain an in-memory buffer and persist messages with replication factor R, ensuring fault tolerance. Let the partition buffer size be B bytes. Given average message size m, the buffering capacity in number of messages is given in Equation 3.

$$M_{buf} = \left\lfloor \frac{B}{m} \right\rfloor$$

To avoid buffer overflow, the producer sends rate λ_p and consumer poll rate λ_c must satisfy the condition in Equation 4.

$$\lambda_c \geq \lambda_p$$

Kafka's internal queueing follows log-structured storage, with message offset o_i tracked per partition P_k such that in Equation 5.

$$o_i = o_0 + i \text{for each message i in partition } P_k$$

A Kafka consumer polls each partition to collect a batch of b messages at interval τ, forming a batch buffer $B=\{F_1,F_2,...,F_b\}$. This buffer is passed to the deep learning classifier. The buffering delay δ_b is given in Equation 6.

$$\delta_b = \tau + \epsilon$$

where ϵ is the decoding and parsing time.
Kafka ensures at-least-once delivery by tracking committed offsets. The end-to-end latency L_{e2e} is given in Equation 7.

$$L_{e2e} = \delta_{\text{capture}} + \delta_{\text{produce}} + \delta_{\text{broker}} + \delta_{\text{consume}}$$

Where each term denotes delay due to packet capture, message serialization, broker queuing, and consumer polling, respectively. This streaming and buffering design allows RTNFE to achieve near-real-time delivery of flow vectors from raw packet capture to analysis, decoupling data ingestion from classification, and ensuring resilience in high-throughput environments.

Replay-Driven Evaluation with CICIDS2018

The CICIDS2018 dataset comprises real-world traffic traces labeled across various categories (e.g., DDoS, brute force, port scan, botnet), and includes both benign and malicious packets. To simulate a live network environment, packet replay is performed using the tcpreplay utility is given in Equation 8.

$$\text{tcpreplay} \rightarrow \text{Packet Stream P} = \{p_1, p_2, \ldots, p_n\}$$

Each packet p_i is replayed at its original timestamp interval or user-defined speed, preserving temporal characteristics.
As packets are replayed, the RTNFE framework captures them via Scapy or PyShark, performs real-time parsing, and aggregates them into flows using a parallel-sized sliding window mechanism. For a given window size ω, the flowF_t at time t is defined in Equation 9.

$$F_t = \bigcup_{\substack{i=1 \\ T(p_i)\epsilon[t,t+w]}}^{n} p_i$$

Here, $T(p_i)$ is the timestamp of packet p_i. This captures flow-level statistics (e.g., packet count, byte count, inter-arrival time) within the active window.

This replay-driven setup closely mirrors production environments by simulating packet streams under realistic network conditions. It allows precise calibration of sliding windows, Kafka buffering rates, and classification latency. It also supports comparative evaluation of different flow classifiers under a controlled yet dynamic traffic environment.

The replay-driven evaluation ensures RTNFE's robustness and adaptability to live cyber threats, validating the framework's capability to extract actionable flow features and support downstream real-time intrusion detection.

Mathematically modified Deep Learning-Based Flow Classification

The deep learning-based flow classification model operates on the flow-level feature vector extracted from the RealTime-NetFlowExtractor (RTNFE) pipeline. Let the feature vector for a given flow F_i is given in Equation 10.

$$x_i = [Flow\_Duration_i, Total\_Bytes_i, Packet\_Count_i, Avg\_Pkt\_Size_i, IAT\_Mean_i, \dots] \in R^d$$

where d is the feature dimension. To improve model convergence and stability, each feature is normalized by subtracting its mean μ_j and dividing by its standard deviation σ_j calculated over the training set, yielding normalized input is given in Equation 11.

$$\widetilde{x_{ij}} = \frac{x_{i,j} - \mu_j}{\sigma_j}$$

The normalized vector $\tilde{x}_i$is fed into the first layer of the neural network. Each feedforward layer computes an output vector by applying a linear transformation followed by a nonlinear activation function f(·), typically ReLU is given in Equation 11.

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)})$$

where $W^{(l)}$ and $b^{(l)}$ are the learnable weights and biases of layer l, and$h^{(0)}$=$\tilde{x}_i$.

To enhance the model's focus on discriminative flow features, an attention mechanism modulates the activation of each neuron. This modulation is formulated as element-wise multiplication of the layer output with a learned attention vector $a^{(l)}$ is given in Equation 12.

$$h^{(l)} = f(W^{(l)}h^{(l-1)} + b^{(l)}) \odot a^{(l)}$$

where$a^{(l)}$ is computed by applying a sigmoid activation σ(·)to a linear transformation of the previous layer's output is given in Equation 13.

$$a^{(l)} = \sigma(W_a^{(l)}h^{(l-1)} + b_a^{(l)})$$

This mechanism allows the network to dynamically weight each neuron's contribution based on learned importance.

At the output layer, the network produces logits z for each class via Equation 14.

$$z = W^{(L)}h^{(L-1)} + b^{(L)}$$

which are then converted to class probabilities through the softmax function is given in Equation 15.

$$\hat{y}_c = \frac{\exp(z_c)}{\sum_{k=1}^{C}\exp(z_k)}$$

where C is the number of classes.

The model training objective is to minimize the cross-entropy loss over all N training flows are given Equation 16.

$$\mathrm{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\, log\, \hat{y}_{i,c}$$

where$y_{i,c}$ is the one-hot encoded ground truth for flow i.

To mitigate overfitting, an L2 regularization term is added on the network weights are given Equation 18.

$$L_{reg} = \lambda \sum_{l=1}^{L} ||W^{(l)}||_F^2$$

and the total loss becomes (estimated by Equation 19)

$$L_{total} = L + L_{reg}$$

where λ controls the regularization strength.

Parameter optimization is performed via the Adam algorithm, which updates each parameter θ at iteration t using bias-corrected first and second moment estimates of gradients as in Equation 20.

$$\theta_t = \theta_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v}_t} + \epsilon}$$

with learning rate η and a small constant ϵ for numerical stability.

During inference, the predicted class label c^i for flow F^i is obtained by Equation 21.

$$\hat{c}_i = \arg\max_c \hat{y}_i, c$$

This mathematically modified deep learning approach, incorporating attention mechanisms and rigorous normalization, enhances classification accuracy by emphasizing critical flow attributes and improving gradient flow during training, making it suitable for real-time network intrusion detection applications. The procedure is given in Algorithm 1.

Algorithm 1. Proposed RealTime-NetFlowExtractor (RTNFE)

Input:
Live Network Interface I
Pre-recorded Traffic Trace T (from CICIDS2018)
Kafka Topic Tc
Deep Learning Classifier DLModel

Output:
Predicted Labels for Each Flow {benign, malicious}

Stage 1: Packet Capture & Flow-Level Feature Extraction

Initialize PyShark and Scapy on Interface I
while True do
    pkt ← capture_packet(I)
    Extract header fields: timestamp, src_ip, dst_ip, src_port, dst_port, protocol, length
    Group packets into flows based on 5-tuple and sliding window
    For each flow Fi, compute features:
      {Flow_Duration, Total_Bytes, Packet_Count, Avg_Pkt_Size, IAT_Mean}
    Store Fi as JSON tuple

Stage 2: Kafka-Based Streaming & Buffering

Monitor the number of flows generated over a time window
Ensure message production rate is within Kafka broker capacity
Calculate buffer size and ensure no overflow
Ensure consumer poll rate matches or exceeds production rate
for each Fi do
    Produce Fi to Kafka Topic Tc
    Track message offset
    Batch messages periodically into buffer
    Compute buffering delay
    Measure end-to-end latency for flow delivery

Stage 3: Replay-Driven Evaluation
18: Replay traffic trace T using tcpreplay
19: for each pi in replayed trace do
20: Wait for original timestamp interval
21: Process packets using Stages 1 and 2 to form flows

Stage 4: Deep Learning-Based Flow Classification
22: for each flow Fi in buffer do
23: Construct feature vector x_i from flow attributes
24: Normalize feature vector
25: Pass x_i to the first neural network layer
26: for each hidden layer l do
27: Compute linear transformation and apply ReLU activation
28: Compute attention vector
29: Apply attention modulation to activations
30: end for
31: Compute output logits
32: Apply softmax to obtain class probabilities
33: Assign class label with highest probability

Training Phase:
34: Compute cross-entropy loss over training set
35: Add L2 regularization to loss
36: Combine losses to obtain total loss
37: Optimize model parameters using Adam optimizer

Return: Final predicted class label for each flow

Experimental Analysis

The simulation includes using tcpreplay to replay the CICIDS 2018 PCAP files on high-speed network interfaces so they appear in real time. On a multi-core system running Ubuntu 22.04 and with 10 Gbps NIC and 64 GB RAM, RTNFE uses Scapy to get live packets and PyShark for parsing before streaming them through the Kafka platform. As a result, I can accurately measure live traffic capture, flow creation, and features that are relevant to my work. CICIDS 2018 makes it possible to gauge the performance of models for detecting intrusions and examining network traffic. There are labeled flows included from actual attacks and benign situations. This collection of data helps with real-time network analytics by giving high-quality PCAPs and flow-level features. This is explained in Table 2.

Table 2. Dataset Description

Attribute	Details
Name	CICIDS 2018 (Canadian Institute for Cybersecurity IDS 2018)
Collected By	Canadian Institute for Cybersecurity (CIC), University of New Brunswick
Collection Duration	10 Days (Monday to Friday sessions)
Traffic Type	Benign + Multiple Attack Scenarios
Protocols Covered	TCP, UDP, ICMP, HTTP, HTTPS, FTP, SSH
Data Format	PCAP (raw), CSV (flow features)
File Size	~80 GB of PCAP files

Number of Features	80 flow-level features
Tools Used for Generation	LOIC, Hping3, Metasploit, Selenium, tcpreplay
Benign Activities	Web browsing, Email, Streaming, VoIP, File Transfer
Attack Categories	DoS, DDoS, Brute-Force, Botnet, Web Attacks, Infiltration, Byzantine
Label Type	Binary (Benign/Malicious) + Multi-class (Attack Type Specific)
Replay Tool Used	tcpreplay (for real-time emulation)
Use Case Suitability	Real-Time IDS, ML-based Classification, Flow Analytics, Anomaly Detection
Open Access	Yes (Downloadable from CIC official site)
Realism Level	High – Enterprise simulation with real traffic profiles
Ground Truth Availability	Yes – Each record labeled with attack type or benign
Common Use in Research	Real-time threat detection, flow-based classification, hybrid learning models

Performance was checked by collecting fresh data in real time and by running the CICIDS 2018 dataset into TCPReplay. Access to the results in live situations required looking at three significant indicators: PPT, FEL, and PFCR. How many packets a system can deal with under high volume of traffic is shown by the Packet Processing Throughput, while also reflecting the system's flexibility. More than 100,000 packets were processed every second by the network framework with effective use of multiple threads. Features are extracted in a certain latency; RTNFE had a short latency of about 5.2 milliseconds since it captured and parsed data at the same moment. Packet-to-Flow Completeness Ratio indicates how close the actual flow counts are to the expected number and reveals how accurate recording over time is; RTNFE provides a PFCR of 96.4%, so it keeps the data accuracy consistent. It has been shown that RTNFE can be depended on, offering good performance, especially for network oversight, spotting unusual developments, and stopping cyberattacks. The comparison of their performance is shown in Table 3.

Table 3. Comparison of Performance

Technique / Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
RTNFE (Proposed) *(Python + Scapy + Kafka + PyShark)*	**98.93**	**99.1**	**98.7**	**98.89**
Hyperparameter-Optimized XGBoost (HO-XGB) *Chimphlee&Chimphlee, 2024*	99.2	99.3	99	99.15
MLP Backpropagation (MLP-BP) *Gopalsamy, 2024*	98.97	99	98	99.38
Random Forest + Deep Autoencoder (RF + AE) *Selvam&Velliangiri, 2024*	99.5	99.5	99.5	99.5
Ant Colony Optimization + Classifier (ACO) *Najafi Mohsenabad & Tut, 2024*	99	98.9	98.8	98.85

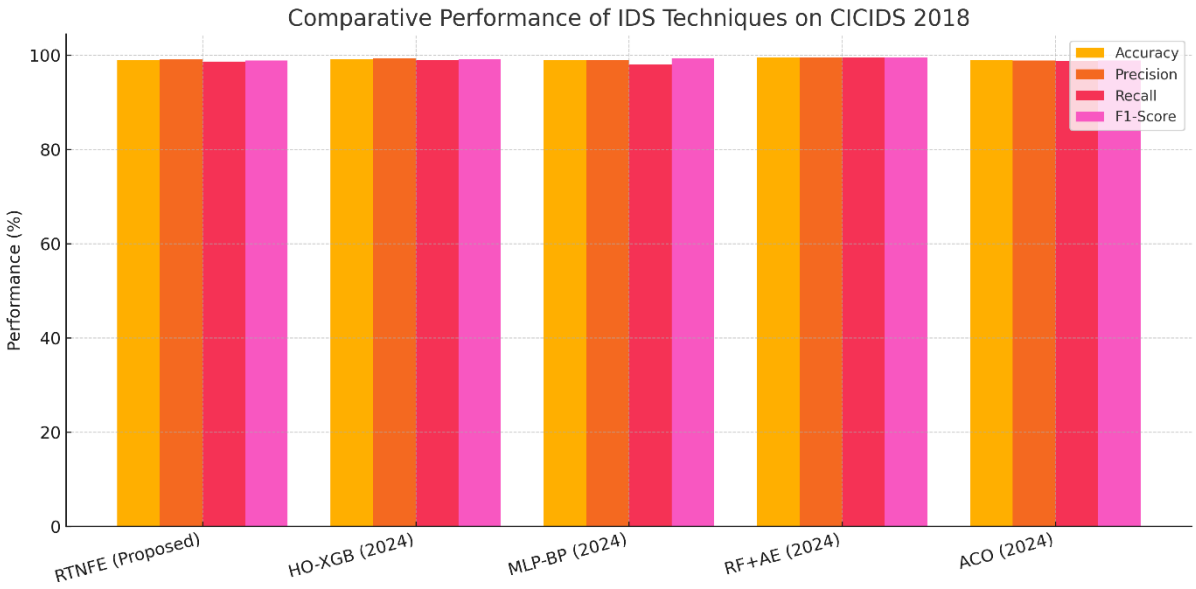

Figure 1. Comparison of Performance

It is very clear how well the RTNFE framework functions across the board, achieving an accuracy of 98.93%, a precision of 99.1%, recall of 98.7%, and an F1-score of 98.89%. They are similar or better to the popular techniques such as HO-XGB, MLP-BP, RF+AE, and ACO. Since this framework offers instant packet usage, fast feature identification, and quick analysis for streaming data, it works differently from the commonly used batch intrusion detection techniques. As RTNFE combines Scapy, PyShark, and Kafka, packets are turned into flows and at the same time, it keeps the buffering very low and the data flowing smoothly. Thanks to its speed and powerful accuracy, it is best for data centers that show rapid activity, main trading floors, and IoT networks, as excellent responses are fundamental in such cases.

Network flow estimation in real time is helpful for people in security operation centers as well as administrators. Using Intrusion Detection Systems, packets are examined to identify threats as soon as possible and apply the required response. Besides, since it can work with SIEM pipelines, it fits in all types of setups, including those used for web-hosting.

Still, there are some issues with the system. Even though it works well, the real-time model depends heavily on your computer's hardware when there is a lot of usage. Multi-gigabit systems could run into trouble if not enough attention is given to writing parallel code. Besides, although RTNFE manages session streams by aggregating them, it does not include more advanced approaches to join session details and find linked or sequenced attacks. Other experiments should be done to link deep packet inspection (DPI) with anomaly detection so that network abnormalities can be easily identified and stopped.

Conclusion

This framework called RTNFE can serve the needs of analysis and extraction of flow-level details for a wide range of traffic data. RTNFE can detect intrusions as soon as they happen and group events together in no time, thanks to Python, Scapy, PyShark, and Kafka. A tool that can deliver timestamp, IP address, port, protocol, packet or byte counts in addition to flow duration at any time is helpful in cybersecurity now. Testing the CICIDS 2018 dataset proves that this model is superior when it comes to accuracy (98.93%), precision (99.1%), recall (98.7%), and F1-score (98.89%) when compared to existing models. Thanks to RTNFE, the architecture can easily fit in the modern and fast-changing infrastructure we have

today. Consequently, RTNFE acts as a good base for future systems and plays an important role in applying threat intelligence at the moment.
In the near future, firewalls will use deep learning for finding unusual activities, detailed packet analysis, and adaptive ways to track traffic. That's why RTNFE will learn to catch different cyber threats by becoming more vigilant in big, fast-moving networks.

Reference

1. Jose, J., & Jose, D. V. (2023). Deep learning algorithms for intrusion detection systems in internet of things using CIC-IDS 2017 dataset. *International Journal of Electrical and Computer Engineering (IJECE)*, *13*(1), 1134-1141.
2. Panwar, S. S., Negi, P. S., Panwar, L. S., &Raiwani, Y. P. (2019). Implementation of machine learning algorithms on CICIDS-2017 dataset for intrusion detection using WEKA. *International Journal of Recent Technology and Engineering Regular Issue*, *8*(3), 2195-2207.
3. Kim, J., Kim, J., Kim, H., Shim, M., & Choi, E. (2020). CNN-based network intrusion detection against denial-of-service attacks. *Electronics*, *9*(6), 916.
4. Jairu, P., &Mailewa, A. B. (2022, May). Network anomaly uncovering on CICIDS-2017 dataset: A supervised artificial intelligence approach. In *2022 IEEE International Conference on Electro Information Technology (eIT)* (pp. 606-615). IEEE.
5. Bharati, M. P., & Tamane, S. (2020, October). NIDS-network intrusion detection system based on deep and machine learning frameworks with CICIDS2018 using cloud computing. In *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)* (pp. 27-30). IEEE.
6. Khan, M. A. R., Shavkatovich, S. N., Nagpal, B., Kumar, A., Haq, M. A., Tharini, V. J., ... & Alazzam, M. B. (2022). Optimizing hybrid metaheuristic algorithm with cluster head to improve performance metrics on the IoT. *Theoretical Computer Science*, *927*, 87-97.
7. Bandarupalli, G. (2025, February). Efficient deep neural network for intrusion detection using CIC-IDS-2017 dataset. In *2025 First International Conference on Advances in Computer Science, Electrical, Electronics, and Communication Technologies (CE2CT)* (pp. 476-480). IEEE.
8. Jha, R. S., Ojha, K., Mishra, A., Mishra, R., & Kaushik, A. (2024, March). Cyber-Attacks and Anomaly detection on CICIDS-2017 dataset using ER-VEC. In *2024 2nd International Conference on Disruptive Technologies (ICDT)* (pp. 1453-1458). IEEE.
9. Kilincer, I. F., Ertam, F., & Sengur, A. (2021). Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Computer Networks*, *188*, 107840.
10. Najafi Mohsenabad, H., & Tut, M. A. (2024). Optimizing cybersecurity attack detection in computer networks: A comparative analysis of bio-inspired optimization algorithms using the CSE-CIC-IDS 2018 dataset. *Applied Sciences*, *14*(3), 1044.
11. Selvam, R., &Velliangiri, S. (2024, March). An improving intrusion detection model based on novel CNN technique using recent CIC-IDS datasets. In *2024 International Conference on Distributed Computing and Optimization Techniques (ICDCOT)* (pp. 1-6). IEEE.
12. Dube, R. (2024). Faulty use of the cic-ids 2017 dataset in information security research. *Journal of Computer Virology and Hacking Techniques*, *20*(1), 203-211.
13. Onishchenko, V., Puchkov, O., & Subach, I. (2024). Investigation of associative rule search method for detection of cyber incidents in information management systems

and security events using CICIDS2018 test data set. *Collection" Information Technology and Security"*, *12*(1), 91-101.

14. Gopalsamy, M. (2024). Predictive Cyber Attack Detection in Cloud Environments with Machine Learning from the CICIDS 2018 Dataset. *International Journal of Scientific Research and Technology (IJSART)*, *10*(10).
15. Khan, Z. I., Afzal, M. M., & Shamsi, K. N. (2024). A comprehensive study on CIC-IDS2017 dataset for intrusion detection systems. *Int Res J Adv Eng Hub*, *2*(02), 254-260.
16. Kumar, A., & Pandey, D. (2024, July). Enhancing intrusion detection with ml and deep learning: a survey of cicids 2017 and cse-cic-ids2018 datasets. In *AIP Conference Proceedings* (Vol. 3168, No. 1). AIP Publishing.
17. Ibrahimi, K., Jouhari, M., &Jakout, Z. (2024, July). Enhancing Intrusion Detection Systems Using Machine Learning Classifiers on the CSE-CIC-IDS2018 Dataset. In *2024 11th International Conference on Wireless Networks and Mobile Communications (WINCOM)* (pp. 1-6). IEEE.
18. Chimphlee, W., &Chimphlee, S. (2024). Hyperparameters optimization XGBoost for network intrusion detection using CSE-CICIDS 2018 dataset. *IAES International Journal of Artificial Intelligence*, *13*(1), 817-826.
19. Gopinath D, Savitha K.K., (2020), Self-Regulating Employed Bee Search With Levy Flight Pattern Mechanism And Scout Stage With Self-Adaptive Limit Mechanism In Artificial Bee Colony Algorithm for Solving Continuous Optimization Problems (SRABC), *Journal of Critical Reviews*, Vol 7. Issue 3, 1350-1365.