National Journal of Antennas and Propagation

ISSN: 2582-2659 Vol. 7, No. 1, 2025 (pp. 253-261)
WWW.ANTENNAJOURNAL.COM

Research Article

# Enhancing Wireless Communication Security using eBPF-Based Packet Filtering and GRU Models in Software-Defined Antenna Networks

**Ali Fadhil Hashim***
*Faculty of Informatics Engineering, Urmia University, Urmia, Iran.*

**KEYWORDS:**

Wireless
eBPF
GRU
Communication
Antennas

**ARTICLE HISTORY:**

| | |
|---|---|
| Received | 12-01-2025 |
| Revised | 22-03-2025 |
| Accepted | 14-04-2025 |

**DOI:**
https://doi.org/10.31838/NJAP/07.01.28

## Abstract

Modern wireless communication systems, particularly those involving smart antennas and software-defined radio networks, require efficient and secure data transmission under increasing cybersecurity threats. Traditional intrusion detection mechanisms often fail to scale with the dynamic demands of high-speed, low-latency wireless environments. This paper proposes an integrated framework that combines Extended Berkeley Packet Filter (eBPF)-based kernel-level packet telemetry with deep learning techniques, specifically Gated Recurrent Unit (GRU) models, for real-time anomaly detection in wireless communication systems. The approach leverages the CSE-CIC-IDS2018 dataset, processed through random sampling, Z-score normalization, and ANOVA-F feature selection, to train a GRU-based detection model capable of analyzing system-level metrics such as TCP session anomalies and malformed packet flows within wireless infrastructures. Packet capture and filtering are performed directly within the Linux kernel using eBPF, ensuring minimal latency. This mechanism is especially suited for software-defined antenna systems, where security must be maintained without compromising throughput. Experimental results show that the proposed system achieves an accuracy of 97.78%, demonstrating its viability for secure and adaptive intrusion detection in next-generation wireless networks.

**Author's e-mail:** Af1432996@gmail.com

**Author's Orcid id:** 0009-0003-4066-8654

**How to cite this article:** Hashim AF, Enhancing Wireless Communication Security using eBPF-Based Packet Filtering and GRU Models in Software-Defined Antenna Networks, National Journal of Antennas and Propagation, Vol. 7, No. 1, 2025 (pp. 253-261).

## Introduction

Prior to the 1990s, packet monitoring and analysis were carried out using the traditional packet filtering mechanism, where all the packets would be duplicated from the kernel space to the user space, thereby creating a packet processing latency. Steven McCanne and Van Jacobson proposed a mechanism known as Berkeley Packet Filter (BPF) between the years 1992-1993, wherein not all the packets are duplicated from the kernel space to the user space [1]. BPF was unlike earlier systems in that it executed programs in a virtual machine designed for register-based processors and included per-application buffers for which no copy of all information was necessary in order to make a decision [2], [3]. eBPF, Extended Berkeley Packet Filter, enables user space applications to provide programs that are executed in the Linux kernel and augment its functionality [4].

Today, DL technology is one of the favorite topics of interest in the fields of machine learning, artificial intelligence, data science, and analytics, as it can learn from input data. Different corporations like Google, Microsoft, etc., actively research it because it can provide remarkable outcomes for various types of problems and classification datasets, and also for regression datasets [5].

An Intrusion Detection System (IDS) monitors network traffic for suspicious activity and alerts when such activity is noted [6]. It is basically a program that inspects a network or system completely or in part and reports any harmful activities [7] .

Conventional IDS tends to fall short in the network security performance requirements, especially for the detection of advanced attacks like DoS and DDoS [8]. The current packet filters are neither flexible nor highly performant. To overcome the shortcomings a Gated Recurrent Units (GRUs) in deep learning models and extended Berkeley Packet Filter (eBPF) for packet filtering are proposed. The GRU model, through its capacity to learn temporal dependencies in sequential data with minimal computational cost, facilitates effective detection of anomalies in network activities, with the help of eBPF, which enables high-performance, tunable in-kernel packet filtering.

## Related Work

This section describes the necessary background on the eBPF and deep learning.

Deokar et al. (2024) [9] performed an empirical analysis to uncover the eBPF application development challenges, spurred by the rapid evolution of the ecosystem and growing usage in the domain of networking and observability. To counteract the absence of a systematic understanding, 743 Stack Overflow posts with the tag “eBPF” were examined by the researchers, with 200 manually classified and the balance further classified using NLP techniques. The precision of the pipeline was 84% with the application of XGBoost across various aspects. The results also stated that 36% of the errors are related to ecosystem primitives and 25% are related to the mistakes of the verifier, reflecting the nuances caused by the mismatches with documentation as well as the rapidly changing tools. Advantages of their methodology are a taxonomy of the development problems and practical recommendations for enhancing tools. Drawbacks are the possible biases with Stack Overflow as the source of data and lower model accuracy with complex categories. The paper is the first large-scale quantitative analysis of eBPF development problems, and the findings provide insight for directing future tooling and documentation.

Gallego-Madrid et al. (2024) [10] Overcome the challenge of implementing intelligent traffic handling in resource-constrained IoT scenarios. They implemented a Multi-Layer Perceptron (MLP) neural network in the Linux kernel using eBPF in order not to rely on ML handling in the user space. By translating an MLP model to C and making the necessary adjustments for eBPF limitations, their implementation saved 97% in execution and 6% in CPU usage over the traditional alternative in the user space. The approach proved stable in a commodity hardware setting, specifically in a 6LoWPAN-based testbed for the purpose of detecting a “Hello Flood” type of attack. The integration minimized latency and resource requirements, demonstrating the possibility of directly deploying ML-enabling security functions within the kernel. Limitations are the complexity involved in implementation because of the eBPF’s substantial verifier limitations, as well as the diminished flexibility in supporting more intricate ML frameworks.

Qiu et al. (2024) [11] solved the problem of Advanced Persistent Threat (APT) detection by introducing a hybrid framework that uses eBPF for the collection of low-level network traffic and a deep learning model based on the Transformer for detection. The system collects kernel-level network traffic and analyzes it with a Transformer to discover APT actions. Tested with a five-server simulated network under different intensities of attacks, the proof of concept achieved a 96.5% detection accuracy and performed better than Snort in terms of accuracy, latency, and resources. Some advantages are high detection accuracy, low latency, and the convenience of supporting complex threat traces. Some disadvantages are dependency on pre-collected features for input to the model and possible complexity with deployment involving kernel-space and user-space reconciliation.

Scholz et al. (2018) [12] Examined the implications of Linux eBPF performance for filtering packets, which surpassed the limitations of traditional centralized firewall configurations. The study offered two case studies: pre-kernel filtering using eXpress Data Path (XDP) and application filtering with socket-attached eBPF. XDP recorded up to 10 million packets per second (Mpps) with just-in-time (JIT) compilation, which was up to 4× faster than iptables and nftables, with socket-level filtering providing fine-grained, application-by-application control without root access. Advantages included enhanced flexibility, lower latency, and decentralized rule management. Drawbacks included the presence of JIT-induced latency outliers and the eBPF program size limitations.

Tolkachova et al. (2023) [13] investigated the application of extended Berkeley Packet Filter (eBPF) technology for extending ransomware detection and monitoring in real-time. Aiming at the limitations of conventional signature-based antivirus solutions, a hybrid detection framework was proposed using eBPF to analyze system calls, process actions, and performance counters. With a secure two-layered virtualized lab, a Support Vector Machine (SVM) classifier, trained across more than 100,000 events, achieved 95.2% accuracy, 94.8% precision, and 95.5% recall for ransomware activity detection. The method is characterized by high speed, flexibility, and rich kernel-space visibility. Issues with implementation are the complexity of development, dependency on hardware, and relative novelty of eBPF's application to ransomware scenarios.

## Methodology

The CSE-CIC-IDS2018 dataset is utilized in the proposed system for IDS. The dataset is used along with IDS and eBPF for packet filtering. The proposed system is thoroughly discussed as given below and as illustrated in Figure 1.

### CSE-CIC-IDS2018

This dataset was created as a collaborative effort by the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC) [11]. It uses a profile-based methodology to produce cybersecurity datasets systematically. The dataset captures rich descriptions of multiple types of intrusions, together with abstract distribution models for applications, protocols, and lower-layer network entities. The dataset contains seven different attack types: Brute-force, Heartbleed, Botnet, Denial of Service (DoS), Distributed Denial of Service (DDoS), Web-based attacks, and internal network penetration. The attacking infrastructure used for the simulation is made up of 50 machines, with the victim organization represented by five departments, with 420 PCs and 30 servers. Each machine on the victim side includes network traffic and log files. The dataset also contains 80 network traffic features extracted with CICFlowMeter-V3, facilitating in-depth traffic analysis with anomaly detection [14].

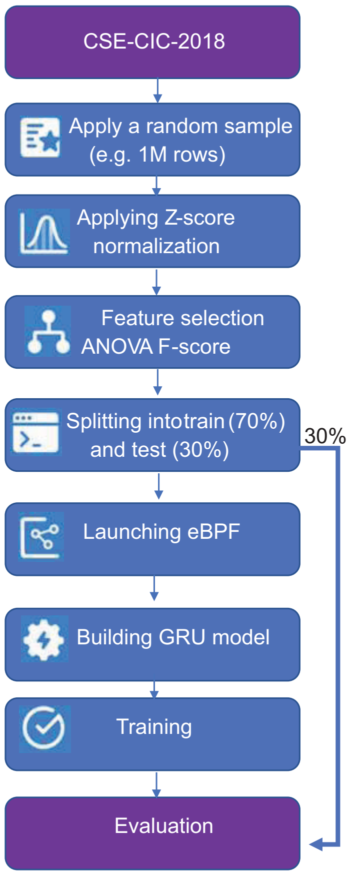

CSE-CIC-2018
Apply a random sample (e.g. 1M rows)
Applying Z-score normalization
Feature selection ANOVA F-score
Splitting intotrain (70%) and test (30%)
30%
Launching eBPF
Building GRU model
Training
Evaluation

**Fig. 1: Proposed system.**

### Random sampling

In this process, each population's member possesses an equal and independent probability of being selected. The sampling frame helps the researcher to choose elements from the population randomly by generating random numbers for each member [15]. In the proposed system, the CSE-CIC-IDS2018 contains 16,232,943 records. Using random sampling, it was decreased to 1,000,0000 with the same class distribution as the dataset, which has 83% normal and 17% attack.

### Z-score normalization

Z-score is an excellent statistical index since it supports the determination of the probability of a given value in a normal distribution, as well as enables a comparison of scores obtained from various normal distributions. This is done through converting (or standardization) the raw scores to a z-score so that the native distribution is translated to a standard normal distribution. The input value is normalized based on Equation 1 as follows [16]:

$$Z(ij) = \frac{aij - \mu}{\sigma}$$

*z*(*ij*) = is the new value.
*aij* =is the old value.
μ =the mean of the column of the input value.
σ =the standard deviation of the column of the input value.

### Anova-F score

Analysis of Variance (ANOVA) is a statistical feature selection algorithm for assessing the significance of numeric features with regard to a categorical target variable. ANOVA is frequently used in classification problems to identify whether the means of two or more groups are statistically different. The F-value in ANOVA measures the ratio of variance between groups to variance within groups. The larger the F-value, the closer the relationship between a feature and the target variable. The score determines whether the means for different classes (as defined by the target vector) are different when single numerical features partition the data [17].

- Steps for ANOVA-based Feature Selection:
  1. Select all features from the original dataset.
  2. Compute the ANOVA F-score between each feature i and the target variable

$$F = \frac{\text{Variance between groups}}{\text{Variance within groups}}$$

$$\text{Variance between groups} = \frac{\sum_{i=1}^{n}\left(\bar{Y}_i - \bar{Y}\right)^2}{(K-1)}$$

$$\text{Variance within groups} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{ni}(Y_{ij} - \bar{Y})^2}{(N-K)}$$

- $K$: Number of groups (or classes)
- $N$: Total number of observations
- ni: Number of observations in group i
- $\bar{Y}_i$: Mean of group i
- $\bar{Y}$: Overall (grand) mean
- $Y_{ij}$: Value of the j, observation in group i

## Data splitting

The dataset is divided into two parts for model training and testing:

1. Training Data (70%): This data is used to train the GRU
2. Incorporate eBPF (Extended Berkeley Packet Filter) monitoring,
3. Testing Data (30%): Assesses the model's performance on new data, ensuring its ability to generalize effectively.

## eBPF (extended Berkeley Packet Filter)

eBPF is utilized to monitor the activity of the system by attaching kernel probes to monitor TCP connect attempts (tcp_connect) and system calls (sys_enter) by process ID. It captures the low-level metrics in using BCC (BPF Compiler Collection) and stores them within hash maps, which facilitates efficient and lightweight monitoring of the system's behavior. The aggregated statistics are then pushed to a connected client, giving insight into system usage and performance. Real-time performance and behavior monitoring is handy in security-sensitive or performance-critical environments.

## GRU (Gated Recurrent Unit)

is employed as a deep learning model to process the time-series or sequential information like the system metrics gathered through eBPF (e.g., TCP connections, syscalls) for discovering the hidden patterns that might signify the regular or malicious activity. The GRU model is trained for 20 epochs, with system metrics (e.g., TCP connections, syscalls, CPU usage) gathered through an eBPF-supporting Linux server. The metrics are captured during training as well as the testing phase, providing visibility of system performance during model training. The trained GRU model is then checked for accuracy, precision, recall, and F1-score over the test corpus. The approach supports the merging of the offline data with live telemetry from the system for enhancing the robustness of the model as well as for facilitating the performance evaluation as per security awareness.

## Evaluation

eBPF gathers metrics such as TCP connections, syscall counts, and CPU usage. Runtime metrics provide a further frame of analysis for assessing not only the accuracy and F1-score of the model, but also the resource usage during the learning process. Once trained, the predictive power of the model is measured by accuracy, precision, recall, and F1 score following regular cybersecurity protocols for evaluation. This combined pipeline enables researchers to test the GRU's detection capacity in realistic monitored conditions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = 2 \times \frac{precision \times Recall}{precision + Recall}$$

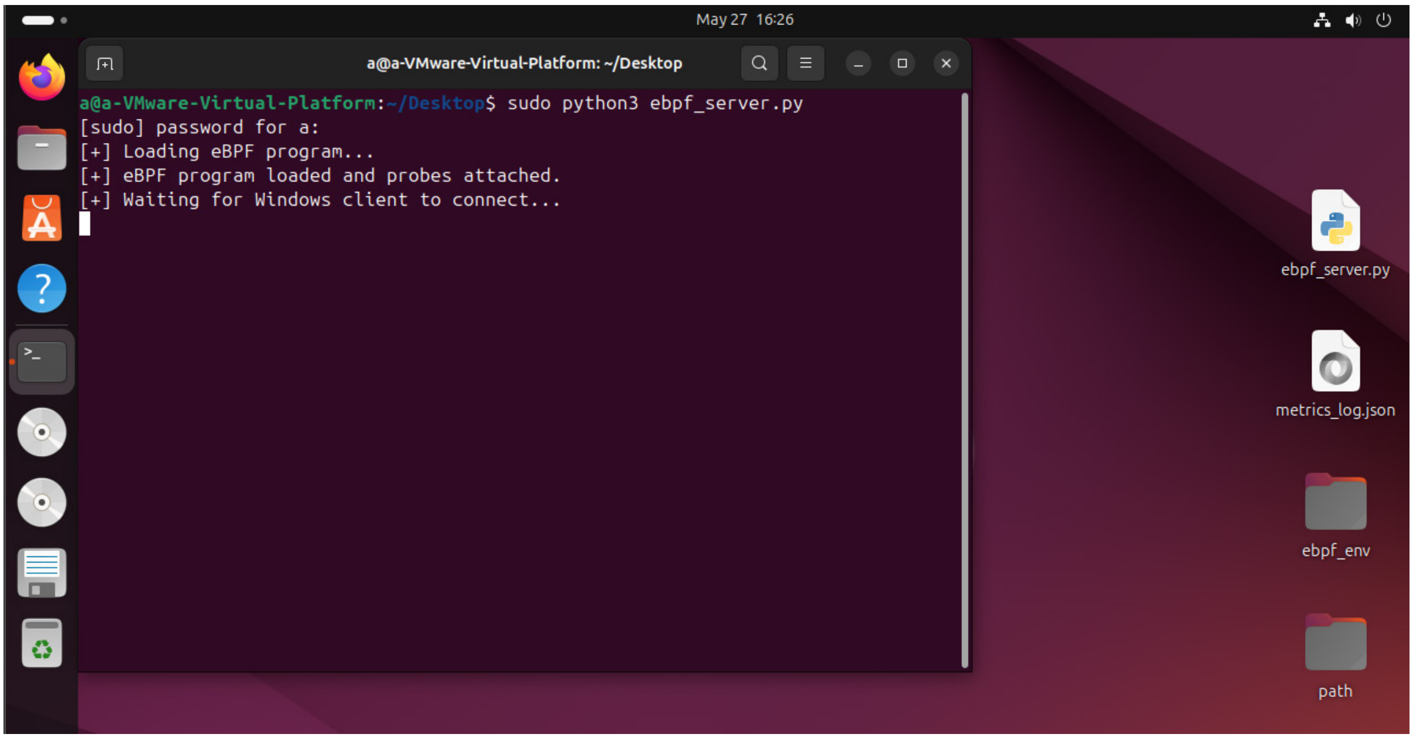
## Implementation and Results

The system proposed employs an IDS via a GRU running on a Windows client (with PyCharm) for model training and a Linux server with Extended Berkeley Packet Filter (eBPF) for monitoring the system. The GRU model for the client is trained with the CSE-CIC-IDS2018 dataset through preprocessing with random sampling, Z-score normalization, and ANOVA F-score-based feature reduction to increase accuracy and efficiency in the detection of abnormally occurring patterns. The eBPF with BCC (BPF Compiler Collection) installed in the Linux server collects kernel-space telemetry, such as TCP connections and system calls, and sends them to the client for analysis. The configuration enables live metrics to be evaluated by the trained GRU model by combining learning with monitoring for efficient, low-latency detection of anomalies.

This shows the Linux server part of the envisioned Intrusion Detection System (IDS) that leverages eBPF for monitoring the system. The terminal reflects the execution required to load kernel-space eBPF programs. The printout attests that the eBPF program had been successfully loaded and its probes (e.g., for monitoring the tcp_connect and sys_enter events) attached successfully to the Linux kernel. The server is now waiting, as reflected in the prompt " Waiting for Windows client to connect.", waiting for incoming requests from the GRU-enabled client running under the Windows operating system. This configuration is commensurate with the system architecture, with the Linux server reading system-level statistics with the aid of eBPF and sending them to the client for anomaly detection with deep learning, leveraging the trained GRU model.

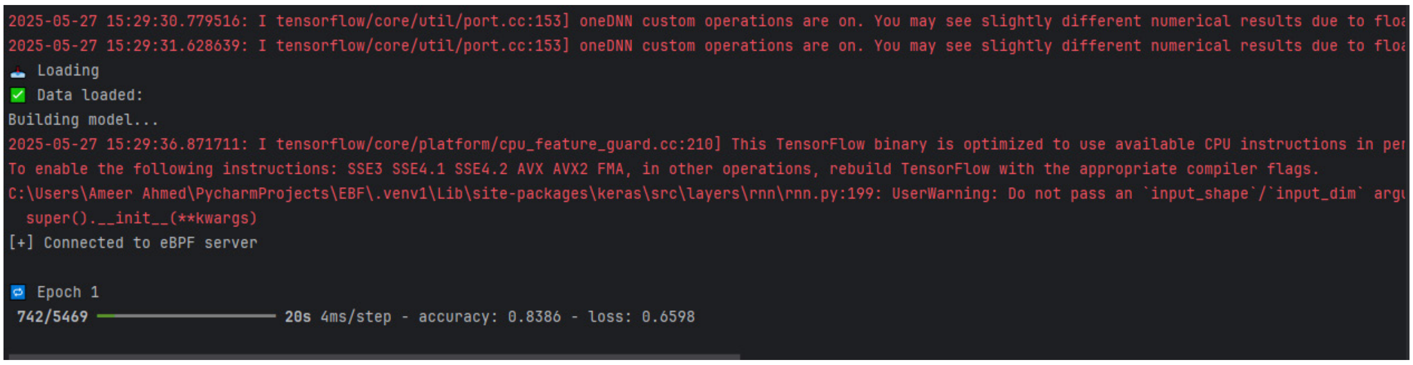
This is the Windows-based client side of the illustrated Intrusion Detection System (IDS), which is running the GRU training process in PyCharm with the help of TensorFlow. The terminal shows successful data loading and model initialization, followed by a "Connected to eBPF server" confirmation message, which affirms an established connection with the Linux server monitoring system metrics through eBPF.

The following is a screenshot of the live telemetry gathered and streamed by the running Linux server, an essential component of the proposed architecture for an IDS. Once the Windows client (192.168.181.1) is successfully connected, the server starts streaming timestamped system indicators in intervals. The indicators are centered around metrics such as tcp_connections, syscalls, cpu_percent, statistics for memory usage, network throughput (bytes_sent_per_sec, bytes_recv_per_sec), and latency (tcp_latency). The measures are obtained via eBPF probes and represent kernel-space monitoring in real-time. The measures are streamed to the client running Windows, where the GRU model is used to process them for the detection of anomalies, a representative application of the practical use of eBPF



```
a@a-VMware-Virtual-Platform:~/Desktop$ sudo python3 ebpf_server.py
[sudo] password for a:
[+] Loading eBPF program...
[+] eBPF program loaded and probes attached.
[+] Waiting for Windows client to connect...
```

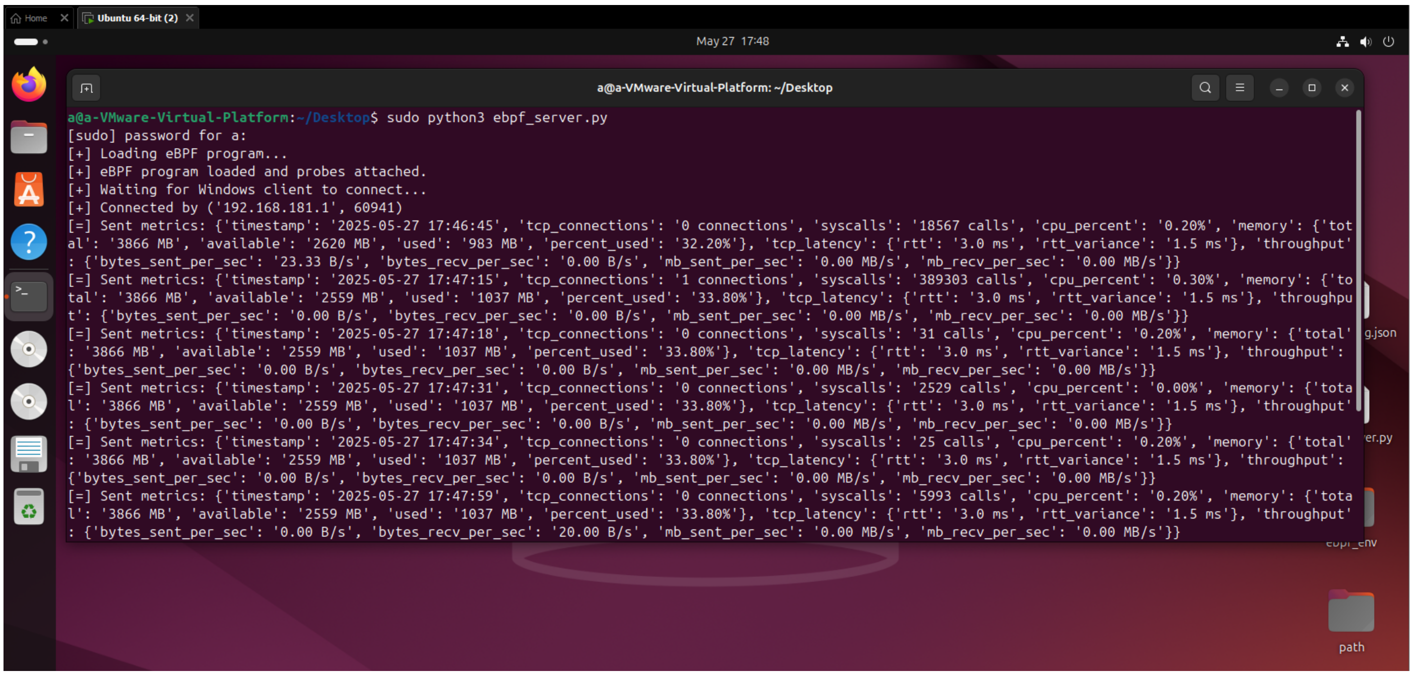
**Fig. 2: Linux server part of the envisioned Intrusion Detection System (IDS).**



```
2025-05-27 15:29:30.779516: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floa
2025-05-27 15:29:31.628639: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floa
📥 Loading
✅ Data loaded:
Building model...
2025-05-27 15:29:36.871711: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in per
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
C:\Users\Ameer Ahmed\PycharmProjects\EBF\.venv1\Lib\site-packages\keras\src\layers\rnn\rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argu
  super().__init__(**kwargs)
[+] Connected to eBPF server

🔁 Epoch 1
 742/5469 ━━━━━━━━━━━━━━━━━━━━ 20s 4ms/step - accuracy: 0.8386 - loss: 0.6598
```

**Fig. 3: Client side of the illustrated Intrusion Detection System (IDS).**



```
a@a-VMware-Virtual-Platform:~/Desktop$ sudo python3 ebpf_server.py
[sudo] password for a:
[+] Loading eBPF program...
[+] eBPF program loaded and probes attached.
[+] Waiting for Windows client to connect...
[+] Connected by ('192.168.181.1', 60941)
[=] Sent metrics: {'timestamp': '2025-05-27 17:46:45', 'tcp_connections': '0 connections', 'syscalls': '18567 calls', 'cpu_percent': '0.20%', 'memory': {'tot
al': '3866 MB', 'available': '2620 MB', 'used': '983 MB', 'percent_used': '32.20%'}, 'tcp_latency': {'rtt': '3.0 ms', 'rtt_variance': '1.5 ms'}, 'throughput'
: {'bytes_sent_per_sec': '23.33 B/s', 'bytes_recv_per_sec': '0.00 B/s', 'mb_sent_per_sec': '0.00 MB/s', 'mb_recv_per_sec': '0.00 MB/s'}}
[=] Sent metrics: {'timestamp': '2025-05-27 17:47:15', 'tcp_connections': '1 connections', 'syscalls': '389303 calls', 'cpu_percent': '0.30%', 'memory': {'to
tal': '3866 MB', 'available': '2559 MB', 'used': '1037 MB', 'percent_used': '33.80%'}, 'tcp_latency': {'rtt': '3.0 ms', 'rtt_variance': '1.5 ms'}, 'throughpu
t': {'bytes_sent_per_sec': '0.00 B/s', 'bytes_recv_per_sec': '0.00 B/s', 'mb_sent_per_sec': '0.00 MB/s', 'mb_recv_per_sec': '0.00 MB/s'}}
[=] Sent metrics: {'timestamp': '2025-05-27 17:47:18', 'tcp_connections': '0 connections', 'syscalls': '31 calls', 'cpu_percent': '0.20%', 'memory': {'total'
: '3866 MB', 'available': '2559 MB', 'used': '1037 MB', 'percent_used': '33.80%'}, 'tcp_latency': {'rtt': '3.0 ms', 'rtt_variance': '1.5 ms'}, 'throughput':
{'bytes_sent_per_sec': '0.00 B/s', 'bytes_recv_per_sec': '0.00 B/s', 'mb_sent_per_sec': '0.00 MB/s', 'mb_recv_per_sec': '0.00 MB/s'}}
[=] Sent metrics: {'timestamp': '2025-05-27 17:47:31', 'tcp_connections': '0 connections', 'syscalls': '2529 calls', 'cpu_percent': '0.00%', 'memory': {'tota
l': '3866 MB', 'available': '2559 MB', 'used': '1037 MB', 'percent_used': '33.80%'}, 'tcp_latency': {'rtt': '3.0 ms', 'rtt_variance': '1.5 ms'}, 'throughput'
: {'bytes_sent_per_sec': '0.00 B/s', 'bytes_recv_per_sec': '0.00 B/s', 'mb_sent_per_sec': '0.00 MB/s', 'mb_recv_per_sec': '0.00 MB/s'}}
[=] Sent metrics: {'timestamp': '2025-05-27 17:47:34', 'tcp_connections': '0 connections', 'syscalls': '25 calls', 'cpu_percent': '0.20%', 'memory': {'total'
: '3866 MB', 'available': '2559 MB', 'used': '1037 MB', 'percent_used': '33.80%'}, 'tcp_latency': {'rtt': '3.0 ms', 'rtt_variance': '1.5 ms'}, 'throughput':
{'bytes_sent_per_sec': '0.00 B/s', 'bytes_recv_per_sec': '0.00 B/s', 'mb_sent_per_sec': '0.00 MB/s', 'mb_recv_per_sec': '0.00 MB/s'}}
[=] Sent metrics: {'timestamp': '2025-05-27 17:47:59', 'tcp_connections': '0 connections', 'syscalls': '5993 calls', 'cpu_percent': '0.20%', 'memory': {'tota
l': '3866 MB', 'available': '2559 MB', 'used': '1037 MB', 'percent_used': '33.80%'}, 'tcp_latency': {'rtt': '3.0 ms', 'rtt_variance': '1.5 ms'}, 'throughput'
: {'bytes_sent_per_sec': '0.00 B/s', 'bytes_recv_per_sec': '20.00 B/s', 'mb_sent_per_sec': '0.00 MB/s', 'mb_recv_per_sec': '0.00 MB/s'}}
```

**Fig. 4: Live telemetry gathered and streamed by the running Linux server.**

for in-kernel monitoring with deep learning inference for cybersecurity under cross-platform deployment.

The given training log shows a well-converged deep learning model over 20 epochs, with stable improvements in both test and training measures. Training loss reduced from 0.1601 to 0.0783, and test loss reduced from 0.1133 to 0.0704, illustrating effective learning and generalization. Correspondingly, training accuracy increased from 95.09% to 97.42%, and test accuracy increased from 96.42% to 97.79%, with no evidence of overfitting and excellent generalization. Convergence is seen at epoch 16, when improvements taper off, supporting the possibility for the application of applying early stopping for reducing the training duration. Overall, the trend is one of stable and well-regularized training, as would be expected following deep learning best practices.

The system performance log across more than 20 epochs is stable in terms of latency with minimal network traffic during model training, with TCP sessions consistently near zero with only short transient spikes at epochs 3, 10, and 17, each with a single session. System call (syscall) activity varies by epoch, with one outlier at epoch 14 with 545,121 calls, which is much above the average of ~7,000 calls per epoch and likely represents a one-off system event or a logging artifact. Throughout the variation, latency is consistently low at 3.0 ms throughout the training process, which suggests that changes in the volume of system calls or TCP sessions did not impact system responsiveness.

Table 1: Training and Testing Performance Metrics Across Epochs for Deep Learning Model.

| timestamp | epoch | Tcp connections | syscalls | latency |
|---|---|---|---|---|
| 5/27/2025 17:14 | 1 | 0 connections | 7886 calls | 3.0 ms |
| 5/27/2025 17:15 | 2 | 0 connections | 6305 calls | 3.0 ms |
| 5/27/2025 17:16 | 3 | 1 connections | 9004 calls | 3.0 ms |
| 5/27/2025 17:16 | 4 | 0 connections | 5208 calls | 3.0 ms |
| 5/27/2025 17:17 | 5 | 0 connections | 6594 calls | 3.0 ms |
| 5/27/2025 17:18 | 6 | 0 connections | 5471 calls | 3.0 ms |
| 5/27/2025 17:18 | 7 | 0 connections | 22387 calls | 3.0 ms |
| 5/27/2025 17:19 | 8 | 0 connections | 5571 calls | 3.0 ms |
| 5/27/2025 17:20 | 9 | 0 connections | 5318 calls | 3.0 ms |
| 5/27/2025 17:21 | 10 | 1 connections | 7607 calls | 3.0 ms |
| 5/27/2025 17:21 | 11 | 0 connections | 4980 calls | 3.0 ms |
| 5/27/2025 17:22 | 12 | 0 connections | 5622 calls | 3.0 ms |
| 5/27/2025 17:23 | 13 | 0 connections | 5170 calls | 3.0 ms |
| 5/27/2025 17:23 | 14 | 0 connections | 545121 calls | 3.0 ms |
| 5/27/2025 17:24 | 15 | 0 connections | 5394 calls | 3.0 ms |
| 5/27/2025 17:25 | 16 | 0 connections | 6489 calls | 3.0 ms |
| 5/27/2025 17:26 | 17 | 1 connections | 5546 calls | 3.0 ms |
| 5/27/2025 17:26 | 18 | 0 connections | 5041 calls | 3.0 ms |
| 5/27/2025 17:27 | 19 | 0 connections | 5435 calls | 3.0 ms |
| 5/27/2025 17:28 | 20 | 0 connections | 5048 calls | 3.0 ms |

Table 2: System-Level Runtime Monitoring Data: TCP Connections, Syscalls, and Latency per Epoch.

| timestamp | epoch | cpu_% | Mem % |
|---|---|---|---|
| 5/27/2025 17:14 | 1 | 0.15 | 34.40% |
| 5/27/2025 17:15 | 2 | 0.1 | 34.30% |
| 5/27/2025 17:16 | 3 | 0.25 | 34.40% |
| 5/27/2025 17:16 | 4 | 0.15 | 34.40% |
| 5/27/2025 17:17 | 5 | 0.15 | 34.30% |
| 5/27/2025 17:18 | 6 | 0.15 | 34.30% |
| 5/27/2025 17:18 | 7 | 0.1 | 34.30% |
| 5/27/2025 17:19 | 8 | 0.1 | 34.30% |
| 5/27/2025 17:20 | 9 | 0 | 34.30% |
| 5/27/2025 17:21 | 10 | 0.2 | 34.40% |
| 5/27/2025 17:21 | 11 | 0.2 | 34.40% |
| 5/27/2025 17:22 | 12 | 0.1 | 34.40% |
| 5/27/2025 17:23 | 13 | 0.25 | 34.40% |
| 5/27/2025 17:23 | 14 | 0.2 | 34.20% |
| 5/27/2025 17:24 | 15 | 0.1 | 34.20% |
| 5/27/2025 17:25 | 16 | 0.1 | 33.90% |
| 5/27/2025 17:26 | 17 | 0.1 | 33.90% |
| 5/27/2025 17:26 | 18 | 0 | 33.90% |
| 5/27/2025 17:27 | 19 | 0.1 | 33.90% |
| 5/27/2025 17:28 | 20 | 0.1 | 33.90% |

Such stability suggests a properly isolated training process with little network dependence and strong system performance consistent with standard procedures for stable machine learning jobs running under dedicated infrastructure.

The CPU usage trace above 20 epochs shows that the model training system had extremely low CPU usage and constant low usage of memory. The CPU usage was extremely low, from 0.0% to 0.25%, which suggests that the task was most likely running on the CPU or it was extremely I/O-bound with minimal CPU process requests. Memory usage was also very much in a low band, from 34.4% to 33.9 % by epoch 20 to 33.9%, which is an even allocation of memory with no leakage or resource contention. The figures testify to a light and well-tuned system configuration for training with a likely single/dedicated or well-separated setting, which is consistent with best practices for efficient deep learning pipelines.

Table 3: CPU and Memory Utilization Metrics per Epoch During Model Execution.

| Accuracy | Precision | Recall | F1_score |
|---|---|---|---|
| 0.9778 | 0.9716 | 0.9778 | 0.9725 |

The final performance results, Accuracy: 97.78%, Precision: 97.16%, Recall: 97.78%, and F1 Score: 97.25%, represent a very effective model with excellent overall performance in classification. The near-tie between precision and recall implies a good balance by the model in keeping false positives and false negatives low. The very high F1 score, the harmonic mean of precision and recall, also ensures excellent performance across both the measures of sensitivity and specificity. The tiny lead for recall over precision can be a good thing when it is preferable to classify rather than misclassify a positive instance, as would be the case in applications for which a missed positive instance is more expensive than a false alarm. These results demonstrate a solid and well-generalized model backed by a well-balanced and well-representative dataset.

## Conclusion

Overall, this study overcame the performance limitations of conventional IDS by integrating GRU and eBPF. The proposed system successfully detected sophisticated network intrusions by leveraging GRU-based deep learning's temporal pattern discovery with the efficient packet filtering capability of eBPF.

The most notable findings indicated that the GRU-eBPF methodology achieved remarkable results with an accuracy of 97.78%, precision of 97.16%, recall of 97.78%, and an F1-score of 97.25%, with strong and stable anomaly detection. The methodology also achieved efficient resource usage, minimal latency, and stable system performance, which highlights its practicability and suitability in actual security situations.

This research is noteworthy in network security because it demonstrates a practical mechanism for coupling deep learning methodologies with kernel-space monitoring frameworks. Future work might investigate further eBPF constraint optimization to support even more sophisticated machine learning designs and test the system in more comprehensive real-world networks to ensure its scalability and efficacy across a wider diversity of attack vectors.

## References

1. A. Sadiq, H. J. Syed, A. A. Ansari, A. O. Ibrahim, M. Alohaly, and M. Elsadig, "Detection of Denial of Service Attack in Cloud Based Kubernetes Using eBPF," *Applied Sciences (Switzerland)*, vol. 13, no. 8, Apr. 2023, https://doi.org/10.3390/app13084700.
2. N. Hedam, *eBPF - From a Programmer's Perspective*. 2023. https://doi.org/10.13140/RG.2.2.33688.11529/4.
3. B. Vinod Kumar and S. Aravind, "Aware and Usage of Information Communication Technology among the MS Ramaiah Institute of Technology, Bangalore: A Study," *Indian Journal of Information Sources and Services*, vol. 9, no. 2, pp. 122–124, May 2019, https://doi.org/10.51983/ijiss.2019.9.2.610.
4. S. Bhat and H. Shacham, "Formal Verification of the Linux Kernel eBPF Verifier Range Analysis," 2022.
5. I. H. Sarker, "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions," Nov. 01, 2021, *Springer*. https://doi.org/10.1007/s42979-021-00815-1.
6. S. Kamolova, H.M. Abbas, D. Abdullayev, C.G, I. Matkarimov, and L. Sachdeva, "Automated disease identification in aquaculture utilizing underwater imaging and YOLOV10 network," *International Journal of Aquatic Research and Environmental Studies*, vol. 5, no. S1, pp. 87–94, Jun. 2025, https://doi.org/10.70102/IJARES/V5S1/5-S1-10.
7. H. J. Hadi *et al.*, "iKern: Advanced Intrusion Detection and Prevention at the Kernel Level Using eBPF," *Technologies (Basel)*, vol. 12, no. 8, Aug. 2024, https://doi.org/10.3390/technologies12080122.
8. A. Naaman, "Machine Learning for Arabic Fake News Detection with Word Embedding During Covid-19 Pandemic," 2022. [Online]. Available: https://www.researchgate.net/publication/364122131
9. M. Deokar, J. Men, L. Castanheira, A. Bhardwaj, and T. A. Benson, "An Empirical Study on the Challenges of eBPF Application Development," in *eBPF 2024 - Proceedings of the ACM SIGCOMM 2024 Workshop on eBPF and Kernel Extensions, Part of: SIGCOMM 2024*, Association for Computing Machinery, Inc, Aug. 2024, pp. 1-8. https://doi.org/10.1145/3672197.3673429.
10. J. Gallego-Madrid, I. Bru-Santa, A. Ruiz-Rodenas, R. Sanchez-Iborra, and A. Skarmeta, "Machine learning-powered traffic processing in commodity hardware with eBPF," *Computer Networks*, vol. 243, Apr. 2024, https://doi.org/10.1016/j.comnet.2024.110295.
11. R. Qiu, H. Luo, S. Jing, X. Li, and Y. Li, "An APT Attack Detection Method Based on eBPF and Transformer," *International Journal of Network Security*, vol. 26, no. 6, pp. 964-972, 2024, https://doi.org/10.6633/IJNS.202411.
12. [12] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance Implications of Packet Filtering with Linux eBPF," in *2018 30th International Teletraffic Congress (ITC 30)*, 2018, pp. 209–217. https://doi.org/10.1109/ITC30.2018.00039.
13. A. Tolkachova, D. Zhuravchak, A. Piskozub, and V. Dudykevych, *Monitoring ransomware with Berkeley Packet Filter (BPF)*. 2023.
14. I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, SciTePress, 2018, pp. 108-116. https://doi.org/10.5220/0006639801080116.
15. T.S. Nanjundeswaraswamy and S. Divakar, "Determination of Sample Size and Sampling Methods in Applied Research," *Proceedings on Engineering Sciences*, vol. 3, no. 1, pp. 25–32, Mar. 2021, https://doi.org/10.24874/pes03.01.003.

16. M. Z. Al-Faiz, A. A. Ibrahim, and S. M. Hadi, "The effect of Z-Score standardization (normalization) on binary input due the speed of learning in back-propagation neural network," *Iraqi Journal of Information & Communications Technology*, vol. 1, no. 3, pp. 42–48, Feb. 2019, https://doi.org/10.31987/ijict.1.3.41.
17. K. Dissanayake and M. G. M. Johar, "Comparative study on heart disease prediction using feature selection techniques on classification algorithms," *Applied Computational Intelligence and Soft Computing*, vol. 2021, 2021, https://doi.org/10.1155/2021/5581806.
18. James, A., Thomas, W., & Samuel, B. (2025). *IoT-enabled smart healthcare systems: Improvements to remote patient monitoring and diagnostics*. Journal of Wireless Sensor Networks and IoT, **2**(2), 11-19.
19. Sadulla, S. (2024). A comparative study of antenna design strategies for millimeter-wave wireless communication. SCCTS Journal of Embedded Systems Design and Applications, 1(1), 13-18. https://doi.org/10.31838/ESA/01.01.03
20. Kavitha, M. (2024). Advances in wireless sensor networks: From theory to practical applications. Progress in Electronics and Communication Engineering, 1(1), 32–37. https://doi.org/10.31838/PECE/01.01.06
21. Sathish Kumar, T. M. (2023). Wearable sensors for flexible health monitoring and IoT. National Journal of RF Engineering and Wireless Communication, 1(1), 10-22. https://doi.org/10.31838/RFMW/01.01.02
22. Antoniewicz, B., & Dreyfus, S. (2024). Techniques on controlling bandwidth and energy consumption for 5G and 6G wireless communication systems. International Journal of Communication and Computer Technologies, 12(2), 11-20. https://doi.org/10.31838/IJCCTS/12.02.02