International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 16, Number 2, pages 99–117, June 2026

Implementation and Evaluation of an FPGA-Based NIPS with Server Response Time–Aware Dynamic Classification Ratio Control

Yuma Ito
Department of Information Sciences, Kogakuin University
Shinjuku, Tokyo, 163-8677, Japan

Ryotaro Kobayashi
Department of Information Sciences, Kogakuin University
Shinjuku, Tokyo, 163-8677, Japan

Received: February 20, 2026
Revised: May 3, 2026
Accepted: June 4, 2026
Communicated by Yasuyuki Nogami

Abstract

In this study, we implement a machine learning–based NIPS on an FPGA to detect and block DDoS attacks, and propose a false-positive reduction method that considers server load. The proposed method periodically measures traffic volume and server response time to estimate the server load. By adopting the larger classification ratio determined from these two metrics, the system performs stepwise blocking control that suppresses unnecessary blocking under low-load conditions while ensuring detection performance under high-load conditions. In the accuracy evaluation, Precision, Recall, F1-score, FPR, and FNR were calculated for each classification ratio, and we quantitatively demonstrated the trade-off whereby Recall and F1-score improve as the classification ratio increases, while FPR also increases. Compared with the previous approach that relies solely on traffic volume as an indicator of server load, the introduction of response time provides a more flexible basis for load-dependent control and suggests the potential to better address low-traffic DDoS attacks under the evaluated conditions. However, in this study, the classification ratio update interval and threshold values are configured as representative examples for system validation. In addition, the response time measurement mechanism is applicable only to TCP traffic. Therefore, the reported results demonstrate the feasibility and effectiveness of the proposed approach within this scope, rather than providing a generalized validation across all attack types and protocols. In the implementation evaluation, a minimum latency of approximately 730 ns was achieved, and it was confirmed that throughput equivalent to approximately 10 Gbps can be processed without packet loss under multiple packet-size conditions. Furthermore, even when the classification ratio update period was changed to 0.5, 1, and 2 s, the latency variation remained below 0.5 %, and 10 Gbps traffic was processed stably, demonstrating the processing stability against changes in the update period.

Keywords: NIPS, DDoS, False Positive

1 INTRODUCTION

In recent years, Internet usage has rapidly increased, accompanied by a growth in the volume of cyberattack traffic. One of the most representative attacks is the Distributed Denial of Service

(DDoS) attack. A DDoS attack primarily aims to impose a high load on Web servers and other services, thereby causing service disruption and degrading user availability.

As a countermeasure against DDoS attacks, the Network-Based Intrusion Detection System (NIDS) has been widely adopted. A NIDS monitors network traffic and detects the presence of anomalous traffic. Many NIDS approaches employ machine learning techniques that are trained in advance to distinguish between normal and malicious traffic. However, since a NIDS performs only detection, it may not be sufficiently effective against attacks such as DDoS attacks that require immediate mitigation. Therefore, the Network-Based Intrusion Prevention System (NIPS), which can take countermeasures against detected malicious traffic, is highly effective.

Nevertheless, a machine learning–based NIPS may degrade user availability by blocking normal traffic when false positives occur. There exists research on DDoS-oriented NIPS that aims to reduce false positives by using traffic volume as an indicator [1]. In our previous work, DDoS attacks were identified based on traffic volume, and the system operated as a NIDS without blocking when no DDoS attack was detected, while operating as a NIPS with blocking under DDoS conditions, thereby reducing false positives in non-attack scenarios.

However, although traffic volume is used in this approach, it does not necessarily enable accurate estimation of server load. Therefore, in this paper, we additionally employ server response time as an indicator, together with traffic volume, to more accurately estimate server load and reduce false positives. By utilizing both traffic volume and response time, we propose a NIPS that operates as a NIDS when the server load is low and as a NIPS when the server load is high.

Since the proposed mechanism mainly targets DDoS attacks, high processing capability is required. It is well known that hardware-implemented NIDS generally achieve higher processing performance than software-implemented NIDS, and FPGA-based NIDS have been extensively studied [2], [3], [4]. Accordingly, to ensure high processing capability, we implement the proposed NIPS on a hardware device, namely an FPGA.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the proposed method. Section 4 presents the experimental setup, and Section 5 reports the results. Section 6 discusses the findings, and finally, Section 7 concludes the paper.

2 RELATED WORK

This section first provides an overview of studies on FPGA-based NIDS and NIPS. Next, the challenges of implementing machine learning on FPGAs are discussed. Furthermore, detection and mitigation algorithms for DDoS attacks and false-positive reduction techniques are reviewed. Finally, the differences between existing studies and this work are clarified, and the position of this study is described.

2.1 FPGA-Based NIDS and NIPS

This subsection discusses the necessity and concerns of implementing NIDS and NIPS on FPGAs.

Zhao et al. implemented a NIPS using an FPGA and a CPU [5]. Their proposed system processes traffic at 100 Gbps and achieves approximately 38× lower power consumption compared with conventional architectures. While software-based NIDS generally face difficulty achieving 100 Gbps processing on a single server, this high throughput was realized by leveraging an FPGA.

Jaic et al. implemented a NIDS using both software and FPGA components [6]. They reported that the open-source NIDS Snort experienced packet drops at 9.1 Gbps, whereas the FPGA-based NIDS processed traffic at 10 Gbps without packet loss.

Essen et al. implemented Random Forest classifiers on CPU, GPU, and FPGA platforms and conducted a comparative evaluation [7]. For a Compact Random Forest consisting of shallow decision trees, the FPGA achieved the highest inference performance. However, for moderately large Random Forest models, the implementation could not fit within a single FPGA due to hardware resource limitations. Furthermore, when decision trees become very deep, techniques that enforce uniform processing time per sample are reported to be impractical from a hardware resource perspective.

These studies indicate that FPGA-based implementations are effective for improving processing performance in high-speed network environments. At the same time, hardware resource constraints must be carefully considered when implementing NIDS or NIPS on an FPGA.

2.2 Implementation of Machine Learning–Based NIDS and NIPS

This subsection reviews studies that implement NIDS and NIPS using machine learning and discusses the associated challenges.

Abdulrezzak et al. conducted a comparative evaluation of signature-based detection and machine learning–based detection in the open-source NIDS Snort [8]. They evaluated multiple attack types, including Probe and DoS attacks, using machine learning classifiers such as Decision Trees, KNN, and Naive Bayes, and compared them with Snort's signature-based approach. While the signature-based method achieved only 25 % detection accuracy, the machine learning classifiers achieved accuracy exceeding 99 %. From these results, they demonstrated that signature-based approaches may fail to detect previously unseen attack patterns, whereas machine learning approaches exhibit high detection capability for such attacks.

However, even with high classification performance, machine learning–based NIDS may still generate false positives and misclassifications [9], [10], [11]. These studies suggest that, although machine learning is more effective than signature-based approaches for detecting unknown attacks, additional mechanisms beyond improving classifier accuracy are required to reduce false positives in practical deployments.

2.3 DDoS Detection Based on Server Response Time

DDoS detection methods that utilize server response delay and temporal variations in traffic have also been reported. Savchenko et al. proposed a method targeting Slow DDoS attacks, in which time-series data such as packet inter-arrival intervals and average delay are used with a time-delay forecasting model to detect attacks [12]. This approach is effective in improving detection accuracy for slow-rate attacks by analyzing temporal changes in traffic statistics.

However, the objective of Savchenko et al.'s study was to improve detection accuracy as an algorithm, and it does not address inline blocking control based on detection results nor dynamic NIPS control aimed at reducing false positives. Moreover, delay information is primarily treated as a statistical feature of traffic flows rather than being directly measured by correlating outgoing packets and corresponding server responses within the NIPS.

2.4 Performance Evaluation Studies Using Server Response Time

Sachdeva et al. evaluated the performance of Web services under DDoS attacks and assessed service quality using metrics such as the average response time and completion rate of HTTP transactions [13]. This study is significant in that it quantitatively demonstrates the impact of DDoS attacks on service quality.

However, in their work, response time is used as a performance evaluation metric and is not directly employed for real-time control or intrusion prevention decision-making at the network device level. In contrast, the fundamental difference of this study lies in utilizing response time not merely as an evaluation metric but as an input parameter for controlling blocking behavior.

2.5 Positioning of This Study

In this study, we implement a NIPS for DDoS attacks using FPGA and machine learning. As discussed in Sections 2.1 and 2.2, the combination of high-speed processing enabled by FPGA and unknown attack detection using machine learning is effective in high-speed network environments; however, false positives caused by misclassification remain a significant challenge in practical deployment. As summarized in Sections 2.3 and 2.4, prior studies utilizing server response time or delay information have mainly focused on improving detection accuracy for slow-rate DDoS attacks or

evaluating service quality under DDoS conditions. However, these studies do not integrate detection results into network-device-level blocking control as a dynamic NIPS control mechanism aimed at suppressing false positives.

This study correlates traffic passing through the NIPS with corresponding server responses to obtain response time and combines it with traffic volume to estimate server load in an inline manner. Based on the estimated server load, the classification ratio of the machine learning decision results is dynamically controlled. As a result, the proposed mechanism is designed to maintain defense performance under DDoS attacks while suppressing unnecessary blocking of normal traffic under non-attack conditions, thereby aiming to balance false-positive reduction and availability preservation.

Furthermore, since the proposed method is implemented on an FPGA, it enables real-time control with low latency and high throughput even in high-speed network environments. Therefore, this study can be positioned as an implementation-oriented NIPS architecture that integrates server-load estimation based on response time with machine learning decisions and explores false-positive reduction through inline dynamic blocking control on FPGA.

To clearly position this study, three levels of comparison are important. First, the baseline traffic-volume-only method [1] controls the NIDS/NIPS behavior using only traffic volume. Its advantage is implementation simplicity, but its load estimation is based on a single indicator. Second, our previous conference paper [14] introduced server response time in addition to traffic volume and demonstrated the basic feasibility of combining these two indicators in an FPGA-based NIPS. Third, the present journal paper extends that foundation by providing a clearer control rationale and a broader implementation-oriented evaluation.

The central difference from the conventional traffic-volume-only method is that the proposed system does not determine the blocking behavior from traffic volume alone. Instead, it estimates server load from both traffic volume and server response time observed inline at the FPGA, and adopts the higher inferred load level to determine the classification ratio. This dual-indicator design is intended to better reflect cases in which server-side overload emerges before traffic volume alone reaches a predefined threshold.

The novelty of the present paper relative to our prior conference version lies in the extended formulation and validation of this control architecture. Specifically, this paper clarifies the availability-security rationale behind the four-level stepwise classification-ratio design, quantitatively characterizes the trade-off among Precision, Recall, F1-score, False Positive Rate (FPR), and False Negative Rate (FNR) for each ratio level, improves reproducibility of model construction by using the publicly available CIC-IDS2017 dataset [15], and evaluates implementation stability for multiple ratio-update periods. Therefore, the contribution of this paper is not a completely new classifier, but an FPGA-based inline NIPS architecture with response-time-aware control and broader experimental validation under the evaluated conditions.

3 PROPOSED METHOD

This paper proposes a false-positive reduction mechanism for a NIPS implemented on an FPGA and enhanced with machine learning to mitigate DDoS attacks. The proposed method reduces false positives by jointly considering traffic volume and server response time. DDoS attacks are primarily intended to impose excessive load on a target server, typically by transmitting a large volume of packets. By deploying a NIPS, malicious DDoS traffic arriving at the server can be detected and blocked. However, there exists a risk that legitimate traffic from normal users may be incorrectly blocked as false positives. Under active DDoS attacks, server protection must be prioritized, even if some false positives are unavoidable. In contrast, during normal operation without DDoS attacks, excessive blocking may degrade service availability for legitimate users. As described in Section 1, systems that adaptively switch between NIDS and NIPS modes based solely on traffic volume have been proposed. In contrast, the proposed system additionally utilizes server response time to estimate server load more directly and to support more reliable load-dependent control for DDoS mitigation.

The proposed mechanism determines whether the system is under a DDoS attack based on two

indicators: traffic volume and server response time. The FPGA operates in NIPS mode when either the traffic volume exceeds the processing capacity of the server or the response time indicates that the server is approaching its processing limit. Both traffic volume and response time are measured every n seconds, and the operating mode (NIDS or NIPS) as well as the classification ratio applied in the subsequent interval are updated according to the measured values.

Among the packets arriving at the NIPS, packets destined for the server are counted every n seconds, and this count is used as the traffic volume indicator. The purpose of measuring server response time is to infer server load from communication latency. The response time is defined as the elapsed time from when a client packet passes through the NIPS to when the corresponding response packet from the server arrives at the NIPS. The transmitted packet and the response packet are associated using the TCP sequence number and acknowledgment number in the TCP header. For each measurement interval of n seconds, one packet transmitted from the client is selected, and its sequence number, source IP address, and TCP payload length are stored. A response packet whose acknowledgment number matches the stored sequence number plus the TCP payload length is identified as the corresponding response. If the TCP payload length is zero, a response packet whose acknowledgment number equals the sequence number plus one is identified instead. The elapsed time between these two events is measured using an on-chip clock counter. Since packet association relies on TCP header information, the proposed mechanism is applicable only to TCP packets.

In the proposed mechanism, the classification ratio is used as a control parameter for adjusting the strength of blocking according to the estimated server load. When both traffic volume and response time indicate low server load, the system prioritizes availability and forwards packets without applying machine learning-based blocking, even if the classifier would identify some packets as malicious. This behavior corresponds to an availability-oriented operating point, where unnecessary blocking of normal traffic is minimized. As the estimated server load increases, the system gradually shifts toward a security-oriented operating point. Three threshold levels are defined for each load indicator, and the classification ratio is increased stepwise from 0/3 to 1/3, 2/3, and 3/3 according to the observed load level. This staged design provides intermediate operating points between pure forwarding and full machine learning-based blocking. Therefore, the proposed mechanism avoids an abrupt binary transition between NIDS and NIPS modes and instead adjusts the blocking strength according to the degree of server load.

The stepwise classification-ratio design is adopted for three reasons. First, it provides a clear and interpretable mapping between server-load levels and blocking strength. Second, it enables gradual control of the availability-security trade-off: lower classification ratios reduce the possibility of false positives, whereas higher classification ratios increase the opportunity to block malicious packets. Third, the discrete ratio levels can be implemented using simple threshold comparisons and selection logic, which is suitable for FPGA-based inline packet processing.

If traffic volume and response time indicate different load levels, the higher load level is adopted to determine the classification ratio. This conservative policy is used because either excessive traffic volume or increased response time can indicate that the server is approaching a high-load condition. In particular, when traffic volume remains low but response time increases, the proposed mechanism can raise the classification ratio based on the response-time indicator, whereas a traffic-volume-only method would continue to regard the situation as low load. The classification ratio is reevaluated and updated every n seconds and is applied during the subsequent interval. These classification ratios and threshold values are introduced as illustrative examples to demonstrate the behavior of the proposed mechanism. They are not optimized or fixed design parameters, and can be flexibly reconfigured according to server characteristics, operational policies, and the selected measurement interval n. Therefore, the evaluation results should be interpreted as a case study of system behavior under representative settings, rather than as a definitive configuration.

An overview of the proposed system is shown in Figure 1, and the flowchart is shown in Figure 2. For incoming packets, traffic volume and response time are measured every n seconds, and the server load is evaluated to update the classification ratio for the next interval. When the estimated load remains low, most or all packets bypass machine learning-based blocking to preserve availability. When the estimated load increases, a larger fraction of packets is subjected to classification, and

packets classified as malicious are blocked. Thus, the proposed mechanism dynamically changes the operating point of the NIPS according to the measured server-load condition. In this study, the entire mechanism is implemented on a physical FPGA device.

From an architectural viewpoint, the proposed FPGA implementation is designed as a streaming packet-processing pipeline. Packets received from the client-side interface first pass through the packet receiver, where header fields required for classification are extracted. The extracted packet-level features are then supplied to the Random Forest classifier, while the same packet stream is also used by the traffic counter and the response-time measurement logic. The Random Forest classifier is implemented as fixed hardware logic composed of comparison and selection circuits corresponding to the trained decision trees. Therefore, packet classification does not require iterative software execution, dynamic memory allocation, or operating-system scheduling for each packet.

The classification-ratio controller is separated from the per-packet forwarding path. It updates the classification ratio only once every measurement interval based on traffic volume and response time, whereas the pass/drop decision for each packet is performed by simple selection logic using the currently stored ratio. Consequently, the control update path and the packet-forwarding path are not serialized. This separation is important for maintaining low latency, because changes in the classification ratio do not require reconfiguration of the packet-processing pipeline. The response-time measurement logic also operates in parallel with the main packet-forwarding path. The ACK matcher observes packets arriving from the server-side interface and compares TCP acknowledgment numbers with stored sequence-number information. This operation is implemented as simple matching logic and does not require software-level packet capture or flow reconstruction. As a result, traffic counting, response-time measurement, machine learning inference, and forwarding decisions are executed concurrently on the FPGA. The low latency and high throughput reported in the implementation evaluation are therefore attributed not merely to the use of an FPGA, but to this streaming and parallel dataflow architecture.

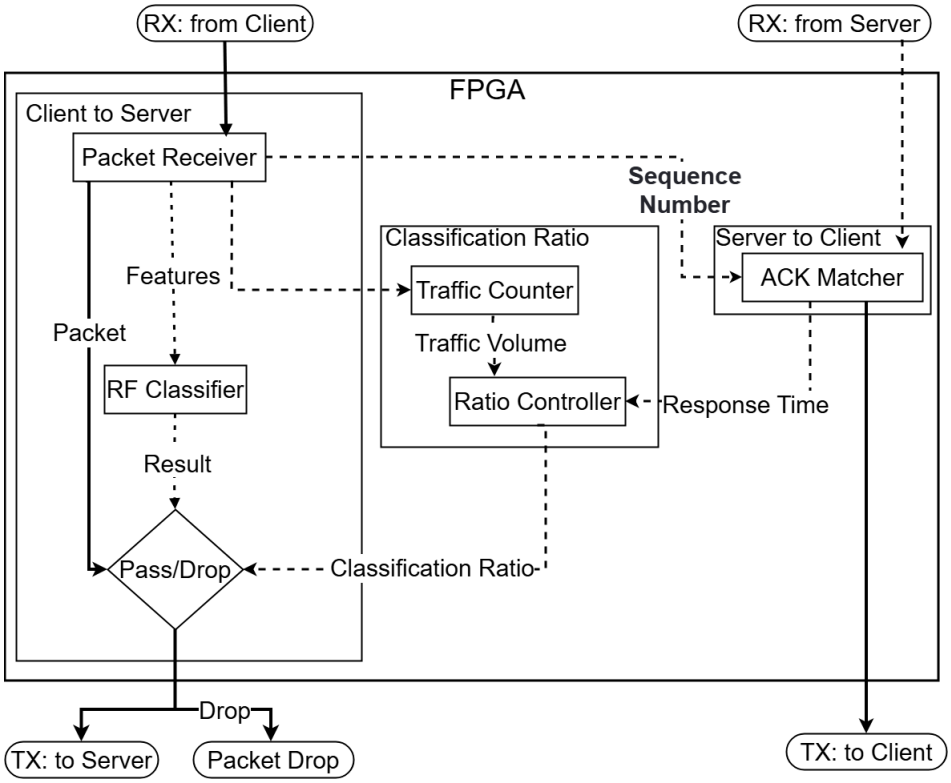

Figure 1: Detailed System Chart

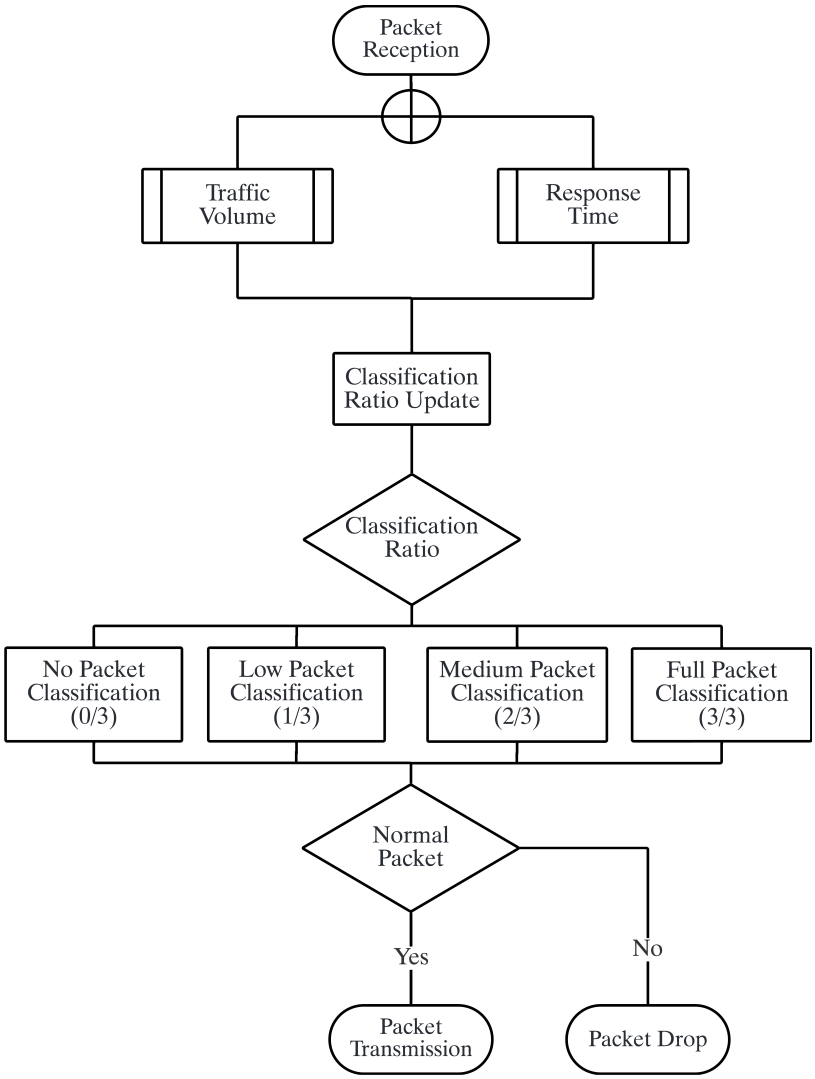

Figure 2: System Flow Chart

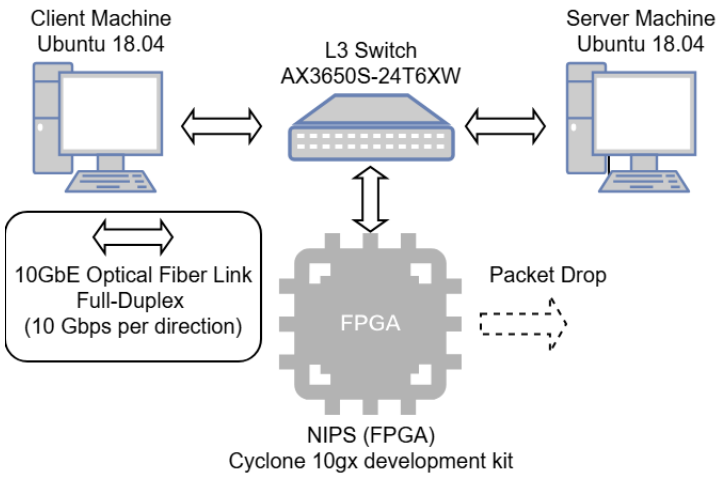

Figure 3: System Environment Configuration Diagram

4 EXPERIMENTS

4.1 Implementation

The experimental environment and FPGA used in the proposed system are described below. The optical fiber links connecting these components support a maximum transmission rate of 10 Gbps for both uplink and downlink directions. The system configuration used in the experiments is illustrated in Figure 3.

- Client machine
  - OS: Ubuntu 18.04-amd64
  - CPU: Intel Core i5-9400 (2.90 GHz)
  - Memory: DDR4-2666 32 GB
- Server machine
  - OS: Ubuntu 18.04-amd64
  - CPU: Intel Core i5-9400 (2.90 GHz)
  - Memory: DDR4-2666 32 GB
- FPGA board
  - Cyclone 10gx Development Kit
  - FPGA: 10CX220YF780E5G
- L3 switch
  - AlaxalA AX3650S-24T6XW

4.2 Machine Learning Algorithm and Dataset

Since the proposed mechanism is implemented on an FPGA, hardware resources are limited, as discussed in Section 2.2. Therefore, implementing computationally complex machine learning algorithms is impractical. In this study, we adopt Random Forest as a machine learning algorithm that is suitable for FPGA implementation while maintaining high classification performance.

A Random Forest consists of multiple decision trees, each of which can be realized using comparison operations and conditional branching. On an FPGA, such a structure can be implemented as a combination of simple `if` statements. In addition, Random Forest is generally known to require relatively low computational complexity and to enable high-speed inference. For these reasons, Random Forest was selected as an appropriate machine learning algorithm for FPGA implementation.

Training was performed offline in a software environment, and a Random Forest classifier consisting of four decision trees with a maximum depth of three is constructed. The branching conditions and threshold values of the trained model are extracted and reproduced as logic circuits on the FPGA.

The features used in this study are four packet-level attributes: source port number, destination port number, packet size, and TCP flags. All of these features can be directly extracted from packet headers and processed without maintaining flow state.

For training, we use the publicly available CIC-IDS2017 dataset [15], which is widely used in intrusion detection research. This dataset includes both normal traffic and multiple types of attack traffic and is designed to emulate a realistic network environment. Although CIC-IDS2017 provides flow-level feature CSV files as well as raw PCAP files, this study does not use the flow-level CSV features. Instead, packets are directly extracted from the PCAP files to construct the training dataset.

This study considers normal traffic, DoS attacks, and DDoS attacks, and performs packet-level feature extraction for a binary classification task. As malicious traffic, 2,148,778 inbound packets

corresponding to DoS and DDoS attacks against a Web server collected on Wednesday and Friday are used. As normal traffic, 4,809,909 outbound packets corresponding to access to an external Web server collected on Monday are used. In total, 6,958,687 packets are used to construct the training dataset, resulting in a class distribution of approximately 69 % normal traffic and 31 % malicious traffic.

Since normal traffic typically dominates in real operational environments, the dataset is configured such that normal traffic constitutes the majority class. However, extremely imbalanced data may introduce training bias and instability in evaluation metrics. Therefore, a sufficient proportion of malicious traffic is included to enable proper evaluation of Precision–Recall characteristics and variations in FPR.

It should be noted that CIC-IDS2017 does not sufficiently include normal external access traffic to the attacked Web server. Therefore, this study allows differences in communication direction when constructing the dataset. However, since the features used in this study are limited to port numbers, packet size, and TCP flags and do not include IP address information, the classification does not depend on specific host identifiers. After extracting features from the dataset and training the Random Forest model, the trained model is implemented on the FPGA.

Since the proposed mechanism measures server response time, the evaluation experiments require an environment in which the server actually returns response packets. If the CIC-IDS2017 packet data are replayed using tools such as Tcpreplay, TCP sessions may not be correctly established, making it difficult to accurately measure response time based on bidirectional communication. Although SYN packets may elicit responses under certain conditions, proper responses to other packet types are not guaranteed. Therefore, in the evaluation experiments, we use self-generated DDoS traffic and actual access traffic to a Web server.

4.3 Classification Performance Evaluation

In this experiment, the classification performance of the proposed method is quantitatively evaluated. The evaluation metrics include Precision, Recall, F1-score, FPR, and FNR. Each metric is defined as follows:

$$\begin{aligned}
\text{Precision} &= \frac{TP}{TP + FP} \\
\text{Recall} &= \frac{TP}{TP + FN} \\
\text{F1} &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\
\text{FPR} &= \frac{FP}{FP + TN} \\
\text{FNR} &= \frac{FN}{FN + TP}
\end{aligned}$$

Here, TP denotes the number of malicious packets that are correctly blocked, FP denotes the number of normal packets that are incorrectly blocked, TN denotes the number of normal packets that are correctly forwarded, and FN denotes the number of malicious packets that are incorrectly forwarded. The evaluation is conducted based on the numbers of blocked and forwarded packets recorded on the FPGA, rather than observations on the server side.

As comparison conditions, three cases are evaluated: (i) without the proposed mechanism, (ii) a method using only traffic volume as an indicator, and (iii) the proposed method using both traffic volume and response time. In the proposed mechanism, the classification ratio is independently determined by the traffic volume indicator and the response time indicator, each taking one of four levels (0/3, 1/3, 2/3, 3/3). The final classification ratio is defined as the larger of the two. Although 16 combinations are theoretically possible, the final classification ratio is reduced to four levels. Therefore, in the experiments, classification performance is evaluated only for these four final classification ratio levels.

The thresholds used in this experiment are shown in Table 1 for the relationship between traffic volume and classification ratio, and in Table 2 for the relationship between response time and classification ratio. These threshold values are set as an example for system validation and can be flexibly adjusted according to the processing capability and operational conditions of each server, since the mechanism is implemented on an FPGA.

The classification ratio update period is set to 1 s. High-load conditions for response time are emulated by introducing artificial delay in the Linux kernel. As normal traffic, approximately 1,000,000 packets corresponding to actual Web browsing traffic to a Web server are used. As malicious traffic, approximately 50,000 packets each are generated using DDoS-Ripper [16] and golang-httpflood [17], resulting in a total of 100,000 malicious packets. The class ratio is set to normal:malicious = 9:1, and the evaluation is conducted on approximately 1,100,000 total packets.

This configuration reflects practical operational environments in which normal traffic dominates and attack traffic is mixed as a minority. In particular, since the objective of this study is to reduce false positives under low-traffic conditions, it is essential to evaluate performance under scenarios where normal traffic is predominant. By ensuring a sufficiently large proportion of normal traffic, variations in FPR can be stably evaluated. Although the proposed mechanism updates the classification ratio at every update period during actual operation, the classification performance in this subsection is evaluated under fixed classification ratios. This design was adopted to isolate the intrinsic trade-off of each ratio setting in terms of Precision, Recall, F1-score, FPR, and FNR. If dynamic ratio updates were enabled during this experiment, the observed results would reflect not only the classification characteristics of each ratio itself but also the effects of control transitions caused by temporal variations in traffic volume and response time. Therefore, fixed-ratio evaluation is first conducted to provide a baseline characterization of the detection-performance trade-off for each ratio level. The time-series behavior of ratio transitions under dynamically changing traffic conditions is outside the scope of this experiment and is discussed as future work.

Table 1: Relationship Between the Number of Packets and the Classification Ratio

Number of Packets	Classification Ratio
$< 40{,}000$	0/3
$< 50{,}000$	1/3
$< 60{,}000$	2/3
$\geq 60{,}000$	3/3

Table 2: Relationship Between the Response Time and the Classification Ratio

Response Time	Classification Ratio
< 20 [μs]	0/3
< 30 [μs]	1/3
< 40 [μs]	2/3
≥ 40 [μs]	3/3

4.4 Latency

This subsection describes the method used to measure the latency of the proposed system. The FPGA was developed using Quartus Prime [18], and its built-in Signal Tap Logic Analyzer was employed for latency measurement. This tool enables waveform acquisition of specified internal signals while the FPGA is operating.

In the measurement, the number of clock cycles is counted from the detection of the Start of Packet (SOP) at the physical-layer receive interface of the FPGA to the output of the End of Packet (EOP) at the transmit interface. The clock frequency used for packet processing on the FPGA is 156.25 MHz.

Although the classification ratio update period is set to 1 s in the accuracy evaluation, latency measurements are conducted with update periods of 0.5, 1, and 2 s to demonstrate the stability of the proposed mechanism.

For latency evaluation, the bit rate is set to 1 Gbps and 10 Gbps, and the packet size is varied among 46, 128, 512, and 850 B. In this paper, the packet size refers to the payload size excluding the Ethernet frame.

During the measurement, packets are continuously transmitted, and latency values are randomly sampled from the ongoing traffic. Latency is measured ten times for each condition.

The commonly used MTU of 1500 B is not evaluated. Since the FPGA equipped with the proposed mechanism does not return responses for packet sizes larger than 850 B, the MTU is set to 850 B in this study.

4.5 Throughput

This subsection describes the method used to measure the throughput of the proposed system. Pktgen-DPDK [19] is used to transmit packets at 10 Gbps, which is the maximum transmission rate of the experimental environment. Pktgen-DPDK is a high-speed packet generator operating on DPDK.

The minimum packet size supported by Pktgen-DPDK is 46 B, and the evaluated packet sizes are 46, 128, 512, and 850 B. To demonstrate stability, throughput is measured with classification ratio update periods of 0.5, 1, and 2 s, as in the latency evaluation.

For each condition, measurements are conducted three times over 60 s, and the presence or absence of packet drops is confirmed. For some packet sizes, 10 Gbps could not be achieved using Pktgen-DPDK; therefore, packets are transmitted at the maximum achievable bit rate in this environment.

Only normal traffic is used for throughput evaluation, and malicious traffic is not included. Since malicious packets are removed from the packet-processing pipeline due to blocking operations, their processing load is smaller, making them unsuitable for throughput evaluation.

As described in Section 4.6, software-based measurement on the server PC cannot reliably observe or respond to all incoming packets under a 10 Gbps environment. Therefore, it is difficult to directly verify whether the FPGA processes 10 Gbps traffic correctly based solely on server-side observations.

Accordingly, throughput is evaluated using a packet counter implemented on the FPGA. The number of packets transmitted by Pktgen-DPDK is compared with the number of packets counted on the FPGA to evaluate throughput.

Since the proposed mechanism measures server response time on the FPGA, response packet processing is required in addition to transmitted packets. The optical fiber links support a maximum transmission rate of 10 Gbps for both uplink and downlink directions, resulting in a theoretical bidirectional processing requirement of up to 20 Gbps. However, accurately measuring the number of response packets from the server under a 10 Gbps environment is difficult; therefore, this experiment evaluates only packets transmitted from the client to the server.

4.6 Preliminary Evaluation of Packet Capture Limitations on the Server PC

This subsection presents a preliminary evaluation to assess the difficulty of software-based packet observation on the server PC under high-throughput conditions. In particular, the objective is to clarify the impact of bit rate and packet size on packet drop rates.

Pktgen-DPDK is used to generate traffic, and the transmission bit rate is gradually varied from 500 Mbps to 10 Gbps. For each bit rate, packet sizes of 46, 128, 512, and 850 B are transmitted for 30 s.

On the receiving side, the number of packets captured at the application level is recorded using `tcpdump`, and packet drop rates at the NIC level are obtained using `ethtool`. The packet drop rates observed on the server PC are summarized in Table 3, and the corresponding results are illustrated in Figure 4.

Table 3: Packet Drop Rate Measured on the Server PC

Bit Rate	Packet Size	Number of packets	NIC Drop Ratio	Kernel Drop Ratio	Tcpdump Capture Ratio
500 Mbps	46	22,449,120	0	0	100
	128	11,432,192	0	0	100
	512	3,434,528	0	0	100
	850	2,129,920	0	0	100
1 Gbps	46	44,629,664	4.37	0	95.63
	128	22,848,896	0.02	0	99.98
	512	6,910,240	0	0	100
	850	4,236,448	0	0	100
3 Gbps	46	134,273,952	71.95	0	28.05
	128	68,127,040	45.15	0	58.85
	512	20,724,384	0.07	36.97	62.96
	850	12,808,576	0	38.37	61.63
5 Gbps	46	224,926,240	83.05	0	16.95
	128	114,101,792	67.28	0	32.72
	512	34,498,080	1.03	60.12	38.85
	850	21,339,488	0.05	61.49	38.46
10 Gbps	46	447,421,600	91.53	0	8.47
	128	225,636,544	83.78	0	16.22
	512	66,787,168	45.45	34.88	19.67
	850	42,750,656	11.40	69.06	19.54

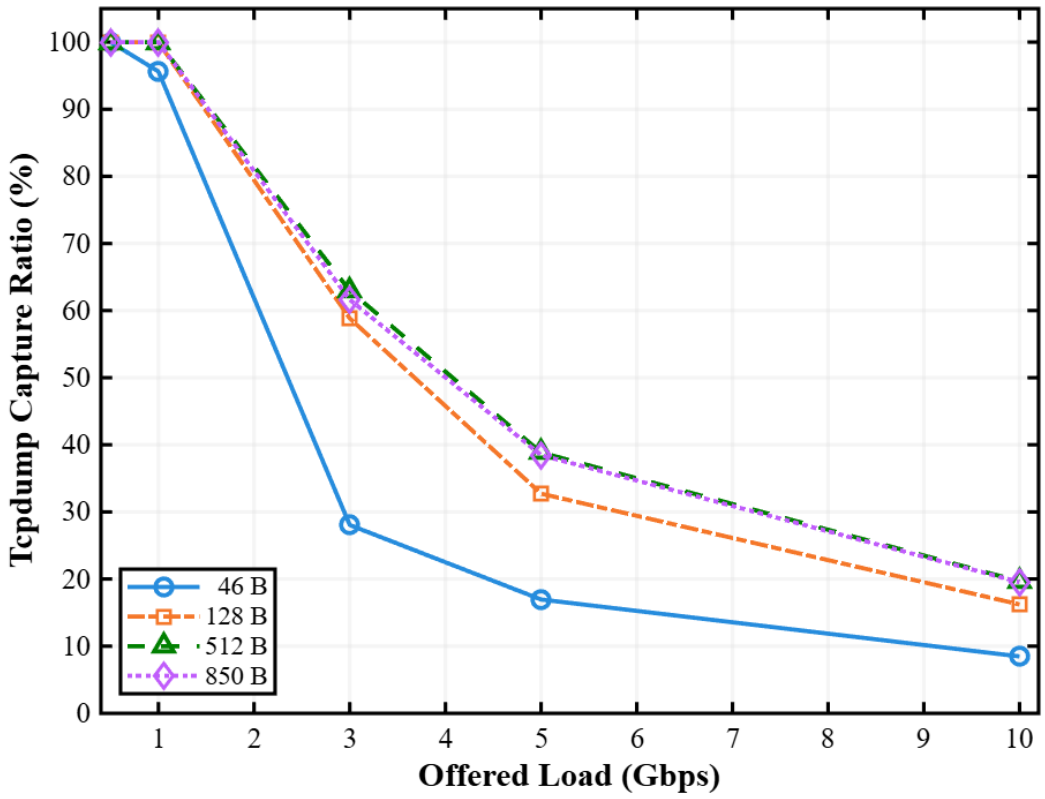

Figure 4: Packet Drop Ratio on the Server PC Measured by tcpdump and ethtool

The results indicate that, in this experimental environment, the server PC is unable to observe and respond to all incoming packets when the bit rate exceeds approximately 1 Gbps.

4.7 Hardware Resource Utilization and Power Consumption

This subsection describes the method used to measure hardware resource utilization and power consumption of the FPGA implementing the proposed system.

The FPGA resource utilization and estimated power consumption are obtained from the Flow Summary and Power Analyzer Summary of the Compilation Report after synthesis and place-and-route in Quartus Prime Pro 20.1 (Build 177).

The power consumption values are estimated under the following conditions: a clock frequency of 156.25 MHz, a standard operating temperature of 25°C, and the default toggle rate settings. It should be noted that these values are estimates provided by the tool and not direct hardware measurements.

5 RESULTS

This section presents the results obtained from the evaluations described in the previous section.

5.1 Classification Performance Results

The classification performance for each classification ratio is summarized in Table 4. Figure 5 illustrates the relationships between the classification ratio and (a) FPR, (b) Recall and FPR, and (c) F1-score. The Precision–Recall curve is presented in Figure 5(d).

When the classification ratio is 3/3, all incoming packets are evaluated by the machine learning classifier, resulting in the same performance as the case without the proposed mechanism. In this case, the accuracy for 1,017,230 normal packets is 97.88 %, the accuracy for 105,832 malicious packets is 97.12 %, and the overall accuracy for 1,123,062 packets is 97.81 %.

Figure 5(a) illustrates that the FPR increases monotonically as the classification ratio becomes larger. A similar tendency can be observed for Recall in Figure 5(b), where higher classification ratios lead to improved detection rates. As depicted in Figure 5(c), the F1-score likewise demonstrates a consistent upward trend with increasing classification ratio. In addition, the Precision–Recall curve shown in Figure 5(d) reveals that Recall improves progressively as the classification ratio increases, whereas Precision remains comparatively stable.

Table 4: Classification Performance for Each Classification Ratio

Classification Ratio	Precision	Recall	F1-score	FPR	FNR
0/3	0.000	0.000	0.000	0.000	1.000
1/3	0.827	0.324	0.466	0.007	0.676
2/3	0.825	0.647	0.725	0.014	0.353
3/3	0.827	0.971	0.893	0.021	0.029

5.2 Latency Results

The latency measured ten times under each condition is summarized in Table 5. Table 5 reports the minimum, maximum, and average latency values for each condition. The average latency is converted from clock cycles to time (ns) using the packet-processing clock frequency of 156.25 MHz.

The average latency measured with update periods of 0.5, 1, and 2 s is shown in Table 6. For all packet sizes and bit rate conditions, the variation in latency caused by changing the update period was less than 0.5 %.

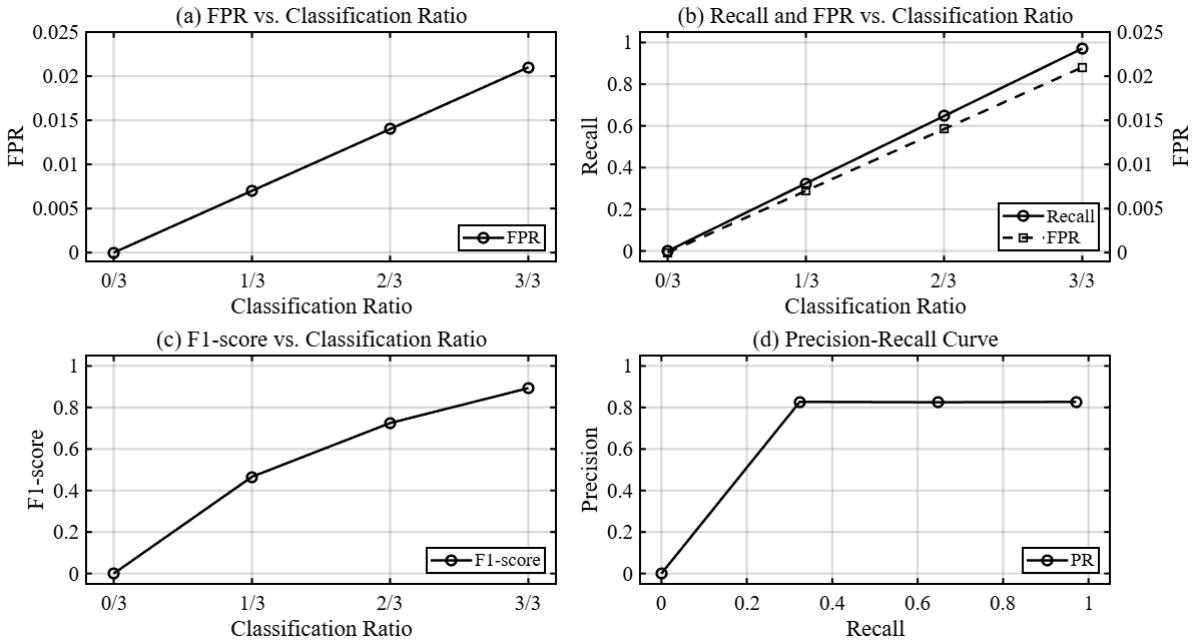

Figure 5: Impact of The Classification Ratio Control on Detection Performance

Table 5: FPGA Latency Measurement Results (in Clock Cycles)

Rate	Size [B]	Min	Max	Ave	Ave Time[ns]
1 Gbps	46	113	115	113.9	729.0
	128	133	135	133.9	857.0
	512	229	231	229.9	1,471.4
	850	313	315	313.9	2,009.0
10 Gbps	46	113	115	114.0	729.6
	128	133	135	134.4	860.2
	512	229	231	230.4	1,474.6
	850	313	315	314.0	2,009.6

Table 6: Impact of Update Period on Latency

Rate	Size [B]	n=0.5	n=1	n=2
1 Gbps	46	114.0	113.9	114.1
	128	134.3	133.9	133.6
	512	229.6	229.9	229.6
	850	313.6	313.9	313.6
10 Gbps	46	114.0	114.0	113.9
	128	134.0	134.4	134.6
	512	230.0	230.4	230.3
	850	314.0	314.0	314.3

5.3 Throughput Results

Throughput is measured three times under each condition, and the results for the update period of 1 s are shown in Table 7. Table 7 presents the number of transmitted packets displayed on the Pktgen-DPDK result screen, the maximum packet rate, and the maximum bit rate calculated from the packet size, rounded to the third decimal place. The bit rate is calculated using the frame size obtained by adding the 14 B Ethernet header to the transmitted packet size. If no packet drop occurs, 'None' is indicated in the Packet Drop column.

Furthermore, Table 8 compares the average maximum bit rates for update periods of 0.5, 1, and 2 s to evaluate the impact of the update period on throughput.

Table 7: FPGA Throughput Measurement Results

Packet Size	Number of packets	Max Packet Rate [PPS]	Max Bit Rate [Gbps]	Packet Drop
46 B	892,809,504	14,962,218	7.18	None
	894,303,232	15,050,680	7.22	None
	900,509,184	14,973,191	7.19	None
128 B	453,201,024	7,564,832	8.59	None
	452,420,640	7,555,204	8.58	None
	455,180,160	7,556,894	8.58	None
512 B	136,993,152	2,285,856	9.62	None
	137,486,016	2,289,847	9.64	None
	138,057,792	2,293,036	9.65	None
850 B	84,961,248	1,420,667	9.82	None
	85,674,816	1,419,330	9.81	None
	85,239,360	1,424,138	9.84	None

Table 8: Impact of Update Period on Throughput

Packet Size	n=0.5 [Gbps]	n=1 [Gbps]	n=2 [Gbps]
46 B	7.19	7.20	7.18
128 B	8.59	8.59	8.59
512 B	9.65	9.63	9.64
850 B	9.82	9.82	9.80

5.4 Hardware Resource Utilization and Power Consumption Results

A portion of the hardware resource utilization obtained from the Flow Summary is shown in Table 9. For items not reported in the Flow Summary, '—' is indicated.

The total power consumption of the implemented system, estimated under the previously described conditions using the Power Analyzer in Quartus Prime Pro 20.1, is 1.39 W.

Table 9: FPGA Resource Utilization

Resource	**Used**	**Available**	**Utilization**
Logic utilization (in ALMs)	12,009	80,330	15 %
Total registers	30,840	—	—
Total block memory bits	2,632,368	12,021,760	22 %
Total RAM Blocks	154	587	26 %

6 DISCUSSION

This section discusses the characteristics of the proposed mechanism based on the experimental results. The objective of this study is not limited to mitigating high-traffic DDoS attacks, but also to suppress false positives while enabling blocking behavior against attacks that impose server load even under low-traffic conditions. To this end, response time is introduced in addition to traffic volume, and the classification ratio is controlled to more directly reflect the server load condition.

In conventional approaches that rely solely on traffic volume, DDoS attacks such as Slowloris, which generate relatively low traffic volume while imposing significant load on the server, may not exceed the traffic threshold and thus may be misclassified as low-load conditions, making blocking behavior less likely to be triggered. In contrast, the response time introduced in this study can reflect increased server load even when traffic volume is small, thereby providing a basis for detecting high-load conditions in scenarios that are difficult to address using traffic volume alone. However, this observation is based on TCP-based response measurements and specific parameter settings, and further investigation is required to generalize this behavior to other protocols and diverse attack scenarios.

The classification performance for each classification ratio (0/3, 1/3, 2/3, 3/3) is presented in Table 4, and the trends of each metric are visualized in Figure 5(a)–(d). As shown in Figure 5(a), FPR is smaller under lower classification ratios, and FPR becomes zero at a classification ratio of 0/3. This result is consistent with the design policy of the proposed mechanism, which aims to minimize unnecessary blocking of normal traffic under low-load conditions. On the other hand, FPR increases monotonically as the classification ratio increases.

Figure 5(b) confirms that Recall improves as the classification ratio increases, while FPR also increases, indicating a trade-off between detection rate and false blocking rate. As shown in Figure 5(c), F1-score increases with higher classification ratios, reflecting the improvement in Recall. Furthermore, the Precision–Recall curve in Figure 5(d) demonstrates that Recall improves as the classification ratio increases, while Precision remains at a relatively stable level.

Since the evaluation dataset has a class ratio of 9:1 (normal to malicious), the Precision–Recall curve is more appropriate than ROC curves for representing classification performance under imbalanced data conditions. The results indicate that the proposed mechanism provides stepwise control of Recall through adjustment of the classification ratio, while Precision remains comparatively stable under the evaluated conditions. However, these results were obtained under fixed classification ratios to isolate the intrinsic performance trade-off associated with each ratio level. Therefore, while the results clarify the baseline relationship between false-positive suppression and detection performance, they do not by themselves represent the time-series behavior of the controller under dynamically changing traffic conditions.

The classification ratio of 3/3 corresponds to evaluating all incoming packets using the machine learning classifier and is therefore equivalent to a standalone machine learning approach. Accordingly, the proposed mechanism achieves staged control, where FPR is reduced under low-load conditions by suppressing the classification ratio, and Recall and F1-score are maintained under high-load conditions by increasing the classification ratio.

Since the proposed method is implemented on an FPGA, high-speed and low-power operation is expected compared with software-based approaches. Existing studies using software-based NIDS such as Snort and Suricata have reported packet drops occurring at several hundred Mbps [20, 21]. In contrast, the latency of the FPGA implementing the proposed system is approximately 730 ns for a packet size of 46 B and approximately 2,000 ns for 850 B packets, and packet processing is sustained without packet drops even under 10 Gbps conditions.

Regarding throughput, processing is sustained at bit rates equivalent to approximately 10 Gbps under multiple packet-size conditions, demonstrating high-speed packet processing capability. Furthermore, latency and throughput are measured under classification ratio update periods of 0.5, 1, and 2 s. The results show no significant change in latency or maximum bit rate when the update period is modified, indicating that the selection of update period does not affect the fundamental packet-processing performance of the FPGA. This suggests that frequent updates of the classification ratio do not destabilize packet processing, demonstrating the processing stability of the proposed

approach.

The estimated power consumption is 1.39 W, indicating a low-power configuration compared with software-based processing. In addition, the FPGA board used in this study retains available logic resources, suggesting that further performance improvement, such as pipeline parallelization, may be achievable in future work.

7 CONCLUSION

This paper proposed an FPGA-implemented NIPS that aims to reduce false positives against DDoS attacks by dynamically controlling the classification ratio based on server load, using server response time in combination with traffic volume. Compared with conventional approaches that rely solely on traffic volume, the proposed method incorporates an additional load indicator based on server response time. As a result, it provides a more flexible basis for determining whether blocking behavior should be activated, particularly in low-traffic environments under the evaluated scope. However, the current implementation has several limitations. First, the response time measurement mechanism is restricted to TCP traffic due to the use of sequence and acknowledgment numbers. Second, the threshold values and classification ratios are configured as illustrative examples for system validation and are not optimized for specific deployment environments. Therefore, the results presented in this paper demonstrate the feasibility and characteristics of the proposed approach under controlled experimental conditions, rather than providing a comprehensive validation across all network scenarios.

This paper presented an FPGA-based inline NIPS architecture that dynamically controls the classification ratio using two server-load indicators: traffic volume and server response time. Relative to the conventional traffic-volume-only method, the present approach expands load estimation from a single indicator to dual indicators and uses the higher inferred load level to determine the aggressiveness of blocking control. Relative to our prior conference report, this journal paper strengthens the contribution through a clearer formulation of the control policy, broader metric-based evaluation, model construction using CIC-IDS2017, and stability analysis across multiple update periods. Accordingly, the contribution of this paper lies in the design and extended validation of response-time-aware inline control on FPGA under the evaluated TCP-based conditions.

In the accuracy evaluation, Precision, Recall, F1-score, FPR, and FNR were calculated for each classification ratio (0/3, 1/3, 2/3, 3/3). The results confirmed that Recall and F1-score increase as the classification ratio increases, while FPR also increases, demonstrating the trade-off between detection performance and false blocking. Through staged classification ratio control, the proposed mechanism provides multiple operating points for adjusting the trade-off between false-positive suppression and detection performance. Lower classification ratios correspond to availability-oriented operation with fewer false positives, whereas higher classification ratios correspond to security-oriented operation with higher detection performance. This design allows the system to change its blocking strength according to the estimated server-load condition, rather than relying on a binary transition between NIDS and NIPS modes.

From an implementation perspective, the proposed method achieved a minimum latency of approximately 730 ns on the FPGA and sustained packet processing without packet drops at throughput equivalent to approximately 10 Gbps under multiple packet-size conditions. Furthermore, when the classification ratio update period was changed to 0.5, 1, and 2 s, latency variation remained below 0.5 %, and packet processing was sustained without packet drops under approximately 10 Gbps conditions. These results indicate that, within the evaluated range of update periods, the update period has little impact on the fundamental packet-processing performance and that the proposed mechanism operates stably when the update frequency is modified.

In addition, the preliminary evaluation in Section 4.6 clarified that software-based observation on the server PC is insufficient under high-throughput conditions, and that evaluation based on FPGA-internal packet counters is effective under 10 Gbps environments. The estimated power consumption was approximately 1.39 W, suggesting that the proposed method can support a high-performance and low-power NIPS implementation.

However, since the proposed mechanism correlates transmitted packets and response packets using TCP sequence and acknowledgment numbers, its applicability is currently limited to TCP traffic. Because DDoS attacks also occur over protocols other than TCP, extending the approach to non-TCP protocols is an important future issue. One possible solution is to generate response-time measurement packets within the NIPS itself.

Furthermore, in the accuracy evaluation presented in this paper, threshold values and classification ratios were configured as an example for system validation, and optimization of thresholds according to server characteristics was not fully explored. In addition, the classification performance was evaluated under fixed classification ratios to isolate the intrinsic trade-off of each ratio level, rather than under time-varying control. In practical operation, however, the classification ratio is dynamically updated according to traffic volume and response time, and acceptable thresholds may vary depending on server operating conditions and load history. Therefore, future work includes dynamic threshold adaptation based on connection states and historical load information, as well as time-series evaluation under real-time classification ratio updates.

References

[1] Y. Ito and R. Kobayashi, *"NIPS Using Machine Learning-Based FPGA that Dynamically Respond to Attack Traffic Volume*," in Proc. Computer Security Symposium (CSS), pp. 1594-1601, 2024 (in Japanese).

[2] M. A. Farooq, A. Rafique, S. A. Fahmy and A. Arora, *"High Throughput Low Latency Network Intrusion Detection on FPGAs: A Raw Packet Approach*," IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 1201-1207, 2025.

[3] S. Pontarelli, G. Bianchi and S. Teofili, *"Traffic-Aware Design of a High-Speed FPGA Network Intrusion Detection System*," IEEE Transactions on Computers, Vol. 62, No. 11, pp. 2322-2334, 2013.

[4] S. Hilgurt, *"A Concise Review of FPGA-Based Hardware Solutions for Network Intrusion Detection*," in Proc. 8th IEEE International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), pp. 164-168, 2021.

[5] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar and J. Sherry, *"Achieving 100Gbps Intrusion Prevention on a Single Server*," in Proc. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 1083-1100, 2020.

[6] K. Jaic, M. C. Smith and N. Sarma, *"A Practical Network Intrusion Detection System for Inline FPGAs on 10GbE Network Adapters*," in Proc. 25th International Conference on Application-Specific Systems, Architectures and Processors, pp. 180-181, 2014.

[7] V. B. Essen, C. Macaraeg, M. Gokhale and R. Prenger, *"Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?*, " in Proc. 20th International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 232-239, 2012.

[8] S. Abdulrezzak and F. A. Sabir, *"An Empirical Investigation on Snort NIDS versus Supervised Machine Learning Classifiers*, " Journal of Engineering, Vol. 29, pp. 164-178, 2023.

[9] K. Rajora and N. S. Abdulhussein, *"Reviews research on applying machine learning techniques to reduce false positives for network intrusion detection systems*, " Babylonian Journal of Machine Learning, Vol. 2023, pp. 26-30, 2023.

[10] T. Saranya, S. Sridevi, C. Deisy, T. D. Chung and M. K. A. A. Khan, *"Performance Analysis of Machine Learning Algorithms in Intrusion Detection System: A Review*, " Procedia Computer Science, Vol. 171, pp. 1251-1260, 2020.

[11] A. Halimaa A. and K. Sundarakantham, *"Machine Learning Based Intrusion Detection System*," in Proc. 3rd International Conference on Trends in Electronics and Informatics (ICOEI), pp. 916-920, 2019.

[12] V. Savchenko, V. Savchenko, O. Laptiev, O. Matsko, I.Havryliuk, K. Yerhidzei and I. Novikova, *"Detection of Slow DDoS Attacks Based on Time Delay Forecasting*, " in Proc. 3rd International Conference on Information Security and Information Technologies (ISecIT), pp. 38-46, 2021.

[13] S. Monika, K. Krishan, S. Gurvinder and S. Kuldip, *"Performance Analysis of Web Service under DDoS Attacks*, " in Proc. 1st International Advance Computing Conference (IACC), pp. 1002-1007, 2009.

[14] Y. Ito and R. Kobayashi, *"Server Load-Aware False-Positive Reduction in an FPGA-Based Machine-Learning Network Intrusion Prevention System*, " in Proc. 13th International Symposium on Computing and Networking Workshops (CANDARW), pp. 245-251, 2025.

[15] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, *"Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization*, " in Proc. 4th International Conference on Information Systems Security and Privacy (ICISSP), pp. 108-116, 2018.

[16] mattiasgeniar, *"DDoS-Ripper*," https://github.com/mattiasgeniar/DDoS-Ripper (Accessed 2026-2-19).

[17] Leeon123, *"golang-httpflood*," https://github.com/Leeon123/golang-httpflood (Accessed 2026-2-19).

[18] Altera, *"Quartus Prime Design Software*," https://www.altera.com/products/development-tools/quartus-prime (Accessed 2026-2-19).

[19] pktgen, *"Pktgen-DPDK*," https://github.com/pktgen/Pktgen-DPDK (Accessed 2026-2-19).

[20] E. Albin and N. C. Rowe, *"A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems*, " in Proc. 26th International Conference on Advanced Information Networking and Applications Workshops (AINAW), pp. 122-127, 2012.

[21] A. A. E. Boukebous, M. I. Fettache, G. Bendiab and S. Shiaeles, *"A Comparative Analysis of Snort 3 and Suricata*, " IEEE IAS Global Conference on Emerging Technologies (GlobConET), pp. 1-6, 2023.