network

MDPI

Article

Lightweight Rescaled Range *R*/*S*-Based Real-Time DDoS Detection for Software-Defined Networks

Mohamad Khattar Awad *, Ghazal Alsholi, Haniah Altabaa, Dania Hani Abu Daqar, Shahad Alshaher and Hamed M. K. Alazemi

Computer Engineering Department, Kuwait University, Sabah Al Salem University City, Al-Shadadiya, Safat, P.O. Box 5969, Kuwait City 13060, Kuwait

* Correspondence: mohamad@ieee.org

Abstract

Software-defined Networking (SDN) is a promising networking architecture that separates the control and data planes to allow flexible network management. However, the SDN architecture makes networks vulnerable to various security threats, such as Distributed Denial-of-Service (DDoS) attacks. A DDoS attack is one of the most common SDN threats, aiming to exhaust a network's computational and bandwidth resources. Self-similarity is a statistical property of time series in which data patterns repeat at different time scales. Several studies have shown that network traffic exhibits increased self-similarity during DDoS attacks, making it a promising tool for DDoS detection. Despite the effectiveness of statistical methods for detecting DDoS, some methods, such as self-similarity, are discarded due to their high computational cost, leading to detection delays. This paper proposes a lightweight Rescaled Range (R/S)-based scheme for effective real-time DDoS attack detection in SDN. The scheme employs the Welford online algorithm to compute statistical parameters of the R/S scheme. Experimental results demonstrate that the proposed scheme efficiently captures changes in self-similarity and detects TCP/UDP DDoS attacks in real time. Moreover, it achieves high detection performance compared to other R/S methods, with a False Positive Rate (FPR) below 0.5% and an average computation time of 0.047 ms.

Keywords: self-similarity; rescaled range analysis (R/S); distributed denial-of-service (DDoS); software-defined networking (SDN); real-time detection

1. Introduction

Legacy networks operate on hardware devices that require manual human configuration [1]. In this architecture, logic is embedded in individual network devices, such as routers and switches, enabling them to process and route packets independently [2]. This design results in rigid, vendor-specific configurations that are difficult to scale and maintain. Attempting to make a minor change in the network requires multiple manual reconfigurations, which are both time-consuming and prone to human error. This static approach also limits the network's agility in optimizing performance, enforcing security policies, and responding to emerging threats, making adaptability to business and consumer demands challenging.

Software-defined Networking (SDN) revolutionized conventional networks by introducing flexibility, programmability, and scalability into their architecture by separating the control plane, which makes decisions about how to operate the network, from the data plane, which forwards packets [3]. This separation transforms networking through

Academic Editor: Andreas Kassler

Received: 16 June 2026
Revised: 23 July 2026
Accepted: 28 July 2026
Published: 5 August 2026

centralized control and abstraction, enabling more flexible, programmable network management suitable for a wide range of applications and services [4]. The evolving complexity of modern networks has underscored the need for scalable, dynamic network management capabilities to respond to changing resource demands and the availability of diverse applications, including smart cities, autonomous vehicles, 5G mobile networks, and the Internet of Things (IoT). Therefore, leading operators such as Amazon, Google, and Microsoft have already started adopting SDN in their cloud data centers [5,6].

OpenFlow is a communication protocol commonly used in SDN. It allows a central controller to directly control network devices. It facilitates an SDN ecosystem with dynamic programmability and network virtualization. It also simplifies and provides a standardized communication protocol for network devices from different vendors [7]. In an OpenFlow-based environment, devices, commonly called vSwitches, maintain a set of flow tables containing flow rules. These flow rules instruct the vSwitches on how to handle incoming packets. Flow rules can be preinstalled or installed on demand by the controller [8]. When a vSwitch receives a packet, it checks the packet header against the existing entries in the flow tables. If no rule matches, the vSwitch sends a PACKET_IN message to the controller, which responds with a FLOW_MOD message prompting the switch to install a new rule for that flow [8]. While this communication channel is essential, it becomes a critical bottleneck during attacks.

The architectural characteristics of SDN have introduced security challenges, making the network vulnerable to various security attacks [9,10]. Attackers exploit the OpenFlow-based control plane by sending a large number of packets that do not match existing flow table rules. Therefore, the switch is forced to send these packets to the controller. The controller then responds by sending new rules to be put into the flow table. This keeps the communication channel between the switch and the controller busy, as shown in Figure 1. Moreover, the controller's computational resources, including the CPU and memory, are exhausted, and the flow tables overflow with new rules. Consequently, the OpenFlow switches become unable to respond to legitimate users [11].

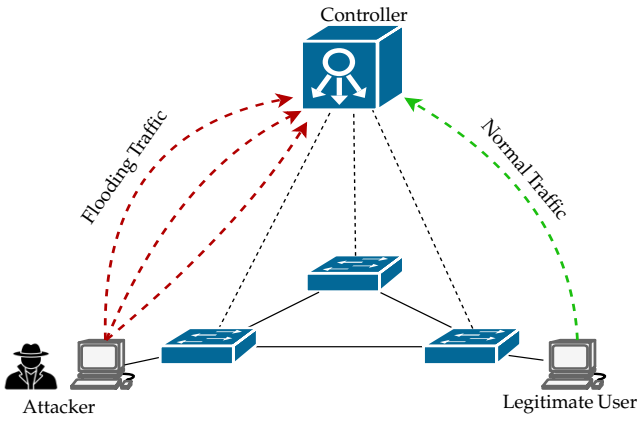

Figure 1. DDoS attack example.

Distributed Denial-of-Service (DDoS) saturation attacks are crucial attacks that aim to exhaust a network's computational and bandwidth resources. DDoS attacks appear in different variants that can target the control plane, the data plane, or both. These include attacks that flood the system, such as Hypertext Transfer Protocol (HTTP) floods, Transmission Control Protocol Synchronize (TCP SYN) floods, User Datagram Protocol (UDP) floods, and Internet Control Message Protocol (ICMP) floods. All these attacks consume significant resources and make it difficult for users to gain access [12]. DDoS attacks can cause serious business disruptions and financial losses by making services inaccessible to users. Attacks

on critical applications, such as government and military systems, can have significant consequences and cause losses. This motivates researchers to develop secure and efficient ways to secure SDN networks against such attacks. In the literature, many researchers have reported various approaches to address this vulnerability and distinguish between normal and attack traffic. Approaches ranging from using traditional intrusion detection systems and priority queues to advanced machine learning and statistical techniques have been proposed in the literature.

Self-similarity is a statistical property that describes how certain characteristics of a network flow remain preserved across different time scales [8]. The Hurst index is a statistical method used to measure the degree of self-similarity of the time series. The value of the Hurst index determines whether the data follows a random walk or has persistent or anti-persistent behavior. In networks, the Hurst index is used to detect saturation attacks by measuring the degree of self-similarity [8]. Early studies by [13] demonstrated that SDN network traffic is characterized by statistical self-similarity and long-range dependence (LRD), a foundational property that remains consistent across various time scales. A growing body of research supports that finding and demonstrates that self-similarity can serve as a predictor of network anomalies. The authors in [14] demonstrate that the network Hurst-index parameter, an indicator of self-similarity, exhibits a measurable increase during attacks, rising from approximately 0.8 to 1.0 during single-mode floods such as ICMP flooding. Other recent related work [8,15–18] has consistently reported similar results.

The Hurst index is typically estimated using several time-domain techniques, such as the Rescaled Range (R/S) analysis, Detrended Fluctuation Analysis (DFA), Absolute Value method, Visibility Graph Analysis (VGA), and Finite Variance Scaling Method (FVSM). In addition, there are frequency-domain techniques, including the Periodogram method, the Abry–Veitch (AV) estimator, and the Whittle estimator [19]. Many studies [15–17] remark that the Rescaled Range (R/S) analysis is one of the most effective methods for DDoS detection. However, it relies on computationally intensive mathematical operations, which may introduce significant delays in real-time DDoS attack detection in SDN environments. Estimating the Hurst index using R/S requires multiple iterative steps and computationally intensive numerical operations, including splitting the time series into multiple segment sizes, calculating the average R/S over multiple window scales, and performing log–log curve fitting and linear regression to estimate the slope.

The authors in [16] highlight that using the Hurst index for anomaly detection is computationally expensive, emphasizing that using the Hurst index for real-time monitoring entails a high computational cost that can increase rapidly, especially as the time window expands. As such, the classical Hurst-index approach is not computationally robust because it requires repeatedly computing the R/S analysis over different-sized windows, corresponding to data segmentation at various scales. This is followed by linear regression analysis and slope estimation. Using the Hurst index for a real-time anomaly detection scenario, a system must wait until enough traffic data is gathered to generate multiple time scales required for a reliable regression analysis. However, by the time enough data is collected to perform the regression analysis, the attacker would have taken down the target server.

This work proposes a robust scheme for computing self-similarity that addresses the computational inefficiencies of R/S-based Hurst-index estimation methods. In this paper, we propose an optimized R/S scheme for real-time DDoS attack detection in SDN. In our scheme, we use fixed-size, overlapping sliding windows of R/S values to monitor traffic self-similarity in real time, without the need for multiple scales or curve fitting. We argue that for the purpose of detecting DDoS attacks, the exact computation of the Hurst index is

unnecessary; instead, we are simply interested in detecting changes in traffic self-similarity. Unlike the Hurst-index computation methods, the proposed scheme eliminates the need for rescaling, regression analysis, and slope computation. As a result, by removing these secondary computational layers, we reduce the algorithm's run time, turning a traditionally slow batch process into a reliable real-time detection approach. In addition, our scheme leverages Welford's online algorithm [20] to streamline the high-complexity operations of the original R/S while maintaining the accuracy of the computed values. Instead of storing and continuously accessing large portions of the dataset, Welford's online algorithm allows efficient, one-pass, incremental computation of the mean and variance without retaining historical samples, making it suitable for real-time environments. We summarize our contributions in the following points:

1. We propose a lightweight and robust self-similarity-based DDoS detector using an optimized R/S for real-time detection in SDN.
2. Our scheme employs a fixed-size sliding overlapping window of R/S to eliminate the computationally expensive Hurst-index estimation while detecting changes in traffic self-similarity accurately.
3. We optimize the R/S computation using Welford's online algorithm to eliminate repeated statistical calculations.
4. We implement the proposed scheme and deploy it in a simulated SDN environment using the Mininet emulator and the Ryu SDN controller.
5. We compare the computational efficiency and detection accuracy of the proposed R/S scheme against Hurst index and other R/S implementations.
6. We demonstrate that our scheme achieves a speedup of 85.3% over other implementations while maintaining accurate DDoS attack detection.

The remainder of this paper is organized as follows: We conduct a comprehensive review of the existing literature on DDoS attack detection and mitigation in Section 2. Then, in Section 3, we provide the background information on the proposed approach. This is followed by a detailed explanation of our proposed scheme and the methods used to develop it, outlined in Section 4. We provide the environment setup details and tools in Section 5. We provide the environment setup details and tools in Section 6. Finally, we summarize the main findings and point out future research directions in Section 7.

2. Related Works

Recent survey papers indicate that DDoS attacks have become one of the most critical and common threats in SDN due to the control plane's centralized architecture [21–23]. This has motivated researchers to develop various approaches to detect and mitigate such attacks in SDN. Survey papers classified the approaches into two categories: machine learning (ML)-based approaches [3,24] and statistical approaches [8,16,25]. In this section, we review the most relevant work in each category.

2.1. ML-Based Approaches

ML is frequently used to detect DDoS attacks due to its efficiency and accuracy in traffic classification. An ML model is usually implemented in the controller to extract hidden patterns from the data and make decisions [2]. Swami et al. [3] developed and evaluated a novel intrusion detection system (IDS) using five different ML classifiers to detect TCP-SYN flooding attacks. Mchina et al. [26] proposed an IDS framework based on a cascaded architecture that combines a CNN-based closed-set classifier with an autoencoder-based zero-day detector. Furthermore, for modern cybersecurity architectures, Tanimu et al. [27] presented a comprehensive analysis of AI-driven threat detection and automated incident response mechanisms. Although the results demonstrate decent performance, the

network's features vary constantly, making it difficult to detect certain types of attacks that the trained models have not seen.

Ahuja et al. [28] proposed a hybrid ML model that concatenates a support vector classifier with RF to classify incoming traffic and detect attacks. The authors generated the training dataset by selecting 23 features that significantly affect detection accuracy. The proposed technique depends on predefined thresholds for the number of packets in the switch and the controller. As with the previous method [3], it remains a challenge to detect new attack types for which the model has not been trained. A combined approach consisting of supervised and semi-supervised classifiers was proposed to address this challenge and effectively detect DDoS attacks in SDN [24]. The supervised classifiers, including K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Naive Bayes, achieved good results in detecting known DDoS attacks but struggled with new attack patterns. On the other hand, the semi-supervised classifiers included Isolation Forest, Basic Auto-encoder, One-Class SVM, and Variational Auto-encoder. The Variational Auto-encoder overcame the previous issue, reaching a precision result of 85%. However, the work found that the optimal time window for detecting saturation attacks varied between the physical and simulated environments. Thus, the achieved results might not be replicable in physical networks.

Graba et al. [29] proposed a two-layer system for DDoS attack detection and mitigation in SDN Internet of Things (IoT) networks. The first layer uses the Signature-based IDS (SNORT) to protect the controller. The second layer consists of four supervised ML models deployed at the controller and trained using cloud services. The authors generated custom TCP SYN flooding traffic for testing and CICDDoS2019 and UNSW-NB15 datasets for validation. The results show that the Decision Tree (DT) model outperformed the Logistic Regression (LR), SVM, and KNN models. The DT achieved detection accuracies of 99.57% for custom-generated traffic and 99.95% and 98.20% in the CICDDoS2019 and UNSW-NB15 datasets, respectively. The controller mitigates that attack by installing flow rules on the switches to block the attacker. Animesh et al. [30] proposed an ensemble machine learning approach utilizing Random Forest (RF) and XGBoost ML models to improve the detection of DDoS attacks in SDN IoT networks. They compared their approach against other ML models, including SVM, non-linear SVM, RF, and Auto-encoder. The ensemble approach outperformed other models, achieving detection accuracy of 99%.

Recently, researchers have started exploring deep learning (DL) techniques for DDoS detection. For example, the authors of [31] introduced a robust DDoS attack detection and mitigation framework based on a deep neural network (DNN). The framework was evaluated using multiple datasets, including the InSDN, CICIDS2018, and Kaggle DDoS datasets. The framework achieved very high detection performance across all datasets, with an accuracy of up to 100%. In addition, the framework can mitigate attacks by blocking malicious traffic, including TCP, UDP, and ICMP, which reduces CPU utilization. In addition, Clinton et al. [32] proposed a convolutional neural network (CNN)-based approach that classifies network traffic as either normal or DDoS. The proposed approach takes network traffic as input, converts it into images, and then uses a CNN to classify them. The proposed approach achieves more than 99% accuracy when tested on the CTU-13 and InSDN datasets and on custom-simulated traffic. However, this approach requires significant computational power and processing time for converting images.

2.2. Statistical Approaches

Statistical methods provide an alternative approach for DDoS detection by analyzing the statistical characteristics of network traffic. Aladaileh et al. [33] investigate in their work the effectiveness of utilizing Shannon entropy for detecting high- and low-rate DDoS

attacks in SDN. In their approach, the entropy is computed from the probability distribution of source IP addresses observed in network traffic to identify abnormal traffic patterns. The results show that their approach achieved lower detection and False Positive Rates in high-rate DDoS attacks than in low-rate DDoS attacks. Shirsath et al. [34] introduced SYNTROPY, a Rényi-entropy-based approach that detects TCP SYN DDoS attacks in SDN. The authors argue that Rényi entropy is more robust than Shannon entropy and can adjust its sensitivity according to network dynamics. The approach calculates Rényi entropy for every time window using features extracted from TCP traffic. An attack is detected when the Rényi entropy exceeds the threshold. The results demonstrate that Rényi entropy outperforms Shannon entropy in detecting DDoS attacks.

Swami et al. [35] proposed a lightweight Interquartile Range (IQR)-based detection method for SYN flooding attacks in SDN. Their method achieves approximately 100% detection accuracy; however, the detection time is 157 s. In [36], the authors introduced a DDoS detection approach for IoT-SDN frameworks based on tracking the frequency of IP packets generated by each device and analyzing their payloads. The method is based on the observation that attack traffic produces packets of the same size, whereas normal traffic produces packets with varying payload sizes. The system compares both metrics and triggers an alarm when either exceeds predefined thresholds. Experimental results show that the framework can detect DDoS attacks with 100% accuracy and a detection time of approximately 2.5 s. However, the approach relies on predefined thresholds that may vary across networks.

Rescaled Range (R/S) analysis was used in a scheme called SA-detector to detect DDoS attacks in SDN [8]. Compared with other state-of-the-art solutions, the SA-detector demonstrates superior accuracy in DDoS detection. However, the study does not provide a direct comparison of the calculation speed. Specifically, the results lack timing data such as detection latency and calculation overhead. In [16], the authors utilized the Hurst index, the auto-regression coefficient, and the coefficient of variation to detect DDoS attacks in conventional networks. The authors compared the performance and accuracy of the Hurst index with other statistical methods under different DDoS attack types. Their results highlight the effectiveness of the Hurst index in providing timely and accurate detection. However, their study was limited to conventional networks and did not explore its application in SDN.

Subsequent work [15] extends the study by utilizing R/S analysis to estimate the Hurst index, and other statistical methods to detect various types of DDoS attacks. The authors recognize Hurst-index estimation via R/S analysis as the most viable method for DDoS detection. However, they exclude it from real-time DDoS testing due to its high computational complexity. Real-time DDoS attack detection in SDN based on the Hurst exponent estimated using the R/S method was proposed in [25]. However, the paper lacks a detailed experimental evaluation, which the authors defer to future work. In our previous work [17], we studied the effectiveness of the Hurst index in detecting DDoS in the controller. We found that attack traffic flows show high levels of self-similarity, whereas normal traffic flows show lower degrees. Moreover, the average difference between the Hurst-index values of normal and attack traffic is approximately 35%, making it effective in classifying traffic flows and detecting attacks.

Although Hurst-index estimation using R/S has been shown to be effective for DDoS detection in previous studies, most work uses it only for offline DDoS detection due to its computational complexity, limiting their experiments to conventional networks, or lacks a detailed analysis of its efficiency and performance in real-time settings.

2.3. Other Approaches

Beyond statistical methods and ML-based approaches, several studies focused on mitigation techniques for SDN saturation attacks. Huang et. al. [37] proposed a low-latency DoS attack detection and mitigation system using port monitoring and priority queues. The system sets a threshold based on baseline measurements recorded during normal network conditions. Although it is effective against basic attacks, this approach depends on the accuracy of the initial traffic classification and may miss attacks that fall below predefined thresholds. Kalash et al. [38] combined firewall rules and IP hashing to detect and mitigate TCP SYN flooding attacks. SDNGuard [39] introduces a probabilistic packet-migration system to defend the controller against attacks. Although it improves legitimate traffic delivery, its reliance on a simple counter for detection and added hardware complexity raises scalability concerns.

The authors of [24] propose vSwitchGuard, a novel framework that effectively detects and counteracts saturation attacks. By combining supervised and semi-supervised machine learning classifiers, vSwitchGuard could detect known and unknown saturation attacks. Also, it detects victim switches and restores them to a normal state by removing fake rules from their flow tables. Besides using OpenFlow 1.5.1 message features in model training, they also used two statistical features of the OpenFlow message payload: the entropy of table-miss packets and the table-miss packet rate, which increased detection accuracy.

Besides the aforementioned work on saturation attack detection and mitigation in SDN, other means include employing rate limiting or load balancing, setting access control policies, leveraging network monitoring tools, and segmenting the network into smaller sub-networks to reduce the impact of an attack, all while maintaining regular firmware and software updates [40].

Although ML-based approaches achieve promising performance and high detection rates, they suffer from several limitations. The main limitations of using ML are the significant computational resources and the time required for training and fine-tuning the model. Model training relies on extensive datasets that represent real-world scenarios, which may be difficult to obtain. Moreover, the model may not detect attacks that were not present in the training dataset. The ML model is deployed in the controller, which creates processing overhead [21,41]. On the other hand, statistical methods are lightweight and do not require training, making them more efficient than ML methods in terms of computational resources. However, some statistical methods are not suitable for real-time DDoS detection because of their computational complexity [42].Hurst-index estimation using R/S analysis requires multiple scaling windows, curve fitting, and slope calculation. This complexity makes the Hurst index unreliable for real-time detection and puts a burden on the controller. Therefore, it is important to propose efficient statistical methods for real-time DDoS detection.

In this paper, we improve our previous work [17] that employs Hurst index for DDoS attack detection. Specifically, we propose a self-similarity-based detector using overlapping sliding windows of R/S, eliminating the need for multiple scaling windows and curve fitting, which consume computational resources and time. Also, we propose the integration of Welford's online algorithm [20] to accelerate the Rescaled Range R/S calculation to achieve effective real-time DDoS detection in SDN. Moreover, we provide a comprehensive experimental analysis on the performance of our enhanced detection scheme. To the best of our knowledge, the integration of Welford's algorithm into the R/S optimization has not been explored in the current literature.

3. Background

This section presents the theoretical foundation necessary for introducing our scheme. Sections 3.1 and 3.2 provide a detailed overview of the Hurst index and the R/S analysis, respectively, and in Section 3.3 we outline Welford's online algorithm, which we used to optimize the R/S calculation.

3.1. Hurst Index

The Hurst index (H) is a statistical method that quantifies the long-term memory of time series data. It indicates whether the time series will either continue its current behavior or revert to its average value over time [43]. The Hurst index is used to measure the self-similarity property in time series. Self-similarity refers to the degree to which a time series exhibits repeated patterns across different scales. It is applied in various domains, such as applied mathematics, queuing theory, chaos theory, etc. [16]. The value of the Hurst index ranges from 0 to 1, where each range corresponds to a different time series behavior [44]:

- $H < 0.5$: The time series values alternate between high and low values (anti-persistent).
- $H > 0.5$: An increase in the data values is likely followed by an increase, and a decrease is likely followed by a decrease (persistent).
- $H = 0.5$: Time series values are not correlated and do not influence future values (random).

The Hurst-index values during normal traffic conditions remain in the range $0.4 < H < 0.6$. This indicates random or uncorrelated traffic behavior. At the onset of abnormal traffic, such as in a DDoS attack, the value of H increases significantly, approaching $H = 1$. High H values during attack traffic indicate persistent traffic behavior. Hence, the Hurst index serves as an effective indicator for the detection of DDoS attacks [15].

There are no analytical methods to calculate the Hurst index; we can only estimate its value through statistical methods. Various statistical methods are used to estimate the Hurst index, including the Rescaled Range R/S analysis method, which was used in [8], Detrended Fluctuation Analysis (DFA), the Wavelet Transform, and the Variogram method; however, this study uses the R/S analysis because of its calculations' simplicity, making it suitable for DDoS detection in networks [45].

3.2. Rescaled Range R/S Analysis

The R/S analysis was introduced by Harold Edwin Hurst during his study of reservoir storage capacity. He used R/S to study the persistence and long-range dependence of the Nile River using time series data [46].

Let the time series be denoted as $X = \{x_1, x_2, \ldots, x_N\}$, where N is the total length of the series. The series is divided into m different scales $\{n_1, n_2, \ldots, n_m\}$, where each n represents a specific segment length. The R/S is computed for each n scale as follows. Usually, segment scales are logarithmically spaced. Divide the series into non-overlapping segments: $K = \lfloor N/n \rfloor$ and for each segment $i = 1, 2, \ldots, K$:

1. Compute the segment mean,

$$\overline{X_i} = \frac{1}{n} \sum_{j=1}^{n} X_{i,j}. \tag{1}$$

2. Construct mean-adjusted series,

$$Y_{i,j} = X_{i,j} - \overline{X_i} \quad \text{for } j = 1, \ldots, n. \tag{2}$$

3. Construct cumulative deviate series,

$$Z_{i,j} = \sum_{t=1}^{j} Y_{i,t} \quad \text{for } j = 1, \ldots, n. \tag{3}$$

4. Calculate range,

$$R_i = \max_{1 \le j \le n} Z_{i,j} - \min_{1 \le j \le n} Z_{i,j}. \tag{4}$$

5. Calculate standard deviation,

$$S_i = \sqrt{\frac{1}{n-1} \sum_{j=1}^{n} (X_{i,j} - \overline{X_i})^2}. \tag{5}$$

6. Calculate Rescaled Range for the segment,

$$(R/S)_i = \frac{R_i}{S_i}. \tag{6}$$

7. After repeating the above steps for all K segments at scale n, compute the average R/S,

$$\mathbb{E}[(R/S)_n] = \frac{1}{K} \sum_{i=1}^{K} (R/S)_i. \tag{7}$$

Steps 1–7 are repeated for all m scales $\{n_1, n_2, \ldots, n_m\}$. Then the power law is used to estimate the Hurst index [16,47],

$$\mathbb{E}[(R/S)_n] \sim C \cdot n^H \quad \text{as} \quad n \to \infty, \tag{8}$$

where H is the Hurst-index value and C is a constant. Taking the log of Equation (8), we obtain the following linear regression equation,

$$\log \mathbb{E}[(R/S)_n] = H \cdot \log n + \log C. \tag{9}$$

From Equation (9), if we perform curve fitting of the $\log \mathbb{E}[(R/S)_n]$ vs. $\log n$ plot, it results in a straight line whose slope is the Hurst index.

The limitation of this Hurst-index estimation is that it requires calculating the average R/S for multiple window scales, followed by log–log curve fitting and slope calculation, which is time-consuming and computationally expensive for real-time traffic. The computational overhead of the classical R/S analysis with non-overlapping windows is controlled by its multi-scale partitioning. While the per-segment operations for the mean-adjustment Equation (2), cumulative summation Equation (3), and standard deviation calculation Equation (5) are linear, $\mathcal{O}(n)$, the primary cost of the classical R/S algorithm stems from its recursive partitioning. For a time series of length N, the algorithm must be executed across m distinct scales. Using the non-overlapping approach, the total time complexity is $\mathcal{O}(Nm) = \mathcal{O}(N \log N)$. The linear regression between $\log(\mathbb{E}[(R/S)_n])$ and $\log(n)$ adds $\mathcal{O}(m)$ time. The combined cost makes the Hurst index time-consuming and inefficient for real-time traffic detection.

To overcome this limitation, we propose using a fixed-size, sliding, overlapping window of R/S that eliminates the need for multiple scales and curve fitting. Figure 2 illustrates the sliding overlapping window approach. Each window has a fixed size and overlaps with the previous window by a predefined step size. Each time the window slides

by the step size, a new $(R/S)_{t+\text{step}}$ value is produced. This overlap enables continuous tracking of self-similarity changes over time, allowing fast response to traffic changes in real time. We emphasize that we are interested in detecting sudden changes in self-similarity rather than accurately estimating the Hurst index. The sliding overlapping window defines the traffic segment used to compute the current window's R/S value. Welford's online algorithm is applied to speed up the R/S computation. As the window advances, Welford's algorithm incrementally updates the mean and standard deviation required for R/S computation, eliminating the need for repeated statistical calculations. The details of Welford's online algorithm are described in the following subsection.

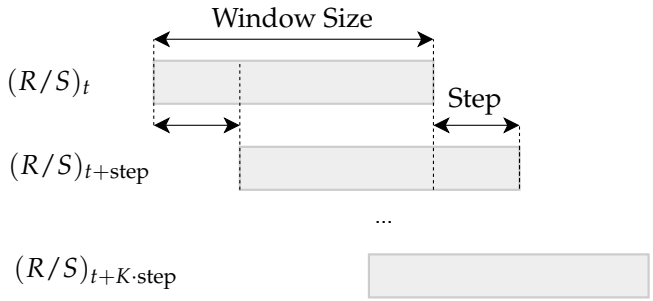

Figure 2. R/S sliding overlapping window. Here t is the current window time and K is the total number of windows.

3.3. Welford's Online Algorithm

Welford's online algorithm was originally proposed by Welford in 1962 [20]. It is an efficient method for calculating the mean and variance in one pass over a dataset by incrementally updating them as new values arrive. This eliminates the need to revisit or store all previous values, which makes this algorithm suitable for streaming data or real-time processing [20,48–50]. Our approach follows the same statistical updating approach introduced by Hanson and West [51,52].

Given a set of n values $x_1, x_2, \ldots, x_n$, the mean $\bar{x}_n$ at step n can be updated iteratively as follows:

$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}, \tag{10}$$

where $\bar{x}_n$ denotes the updated mean after processing the n^{th} sample, $\bar{x}_{n-1}$ denotes the previous mean, x_n denotes the current n^{th} sample value, and n is the total number of samples.

Welford's algorithm incrementally updates the sum of squared differences from the current mean, denoted $M_{2,n}$, as follows,

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n). \tag{11}$$

To simplify the expression, we can define the intermediate quantities as follows:

$$\delta_1 = x_n - \bar{x}_{n-1}; \tag{12}$$

$$\delta_2 = x_n - \bar{x}_n, \tag{13}$$

Using these definitions, the update formula can be written as follows,

$$M_{2,n} = M_{2,n-1} + \delta_1 \cdot \delta_2. \tag{14}$$

Once $M_{2,n}$ is calculated, the biased sample variance σ_n^2 and unbiased sample variance s_n^2 can be computed respectively as follows:

$$\sigma_n^2 = \frac{M_{2,n}}{n}; \tag{15}$$

$$s_n^2 = \frac{M_{2,n}}{n-1}. \tag{16}$$

Welford's algorithm is considered numerically stable [20,48–50], and its incremental nature makes it suitable for online computations, such as analyzing network traffic online. In the following section, we adopt the fixed-size sliding window and the Welford algorithm to optimize the R/S method for online analysis of SDN traffic self-similarity.

4. Proposed Approach

Our proposed scheme follows the same essential steps as the original R/S method, while optimizing how these steps are computed to enhance efficiency without compromising accuracy. Particularly, the implementation steps of our sliding window R/S approach typically follow these steps. First, packets are sniffed continuously from the network and then appended to a buffer, which is usually the size of the sliding window. Once the buffer reaches the predefined window size, the R/S calculations begin. We leverage Welford's algorithm to compute the mean and standard deviation, which are both essential for the R/S calculation of the current window, while the buffer is still accumulating network packets. By the time the buffer reaches the predefined window size, the standard deviation and the range can be computed immediately. Our proposed scheme performs constant-time $\mathcal{O}(1)$ updates to the running statistics using Welford's method, with a total time complexity of $\mathcal{O}(W)$, where W is the window size. The detailed steps of our scheme are presented in Algorithms 1–5. All algorithms are deployed on the controller.

Algorithm 1 Real-time windowing.

```
1:  Input: window_size, step_size
2:  Initialize: packet_buffer ← [ ]
3:  next_end ← current time + window_size
4:  procedure PACKETSNIFFER
5:      while True do
6:          (t, l) ← capture packet                ▷ t: arrival time, l: packet length
7:          append(packet_buffer, (t, l))
8:      end while
9:  end procedure
10: procedure WINDOWSCHEDULER
11:     while True do
12:         if current time ≥ next_end then
13:             start ← next_end - window_size
14:             PROCESSWINDOW(start, next_end)
15:             next_end ← next_end + step_size
16:         end if
17:     end while
18: end procedure
```

Algorithm 1 summarizes the steps of the real-time packet windowing mechanism. Initially, the algorithm receives the window size and step size as inputs and initializes a packet buffer to collect packets. The packet sniffing routine PACKETSNIFFER continuously captures incoming packets using the `Scapy v2.5.0` tool (a Python library for packet generation and manipulation). (Lines 4–9). We extract each packet's length and timestamp and

append it to the packet buffer (Lines 6–7). At the time, the scheduling routine, which is called WINDOWSCHEDULER, manages the time-based windowing for R/S computations as described in (Lines 10–18). The algorithm checks whether the current time window has reached the end of the active window (Line 12). If the condition is true, the buffered packet data corresponding to that window is forwarded to the PROCESSWINDOW procedure (Line 14). After that, the window end time is advanced by the predefined step size to enable continuous real-time processing (Line 15).

Algorithm 2 ProcessWindow: Binning and normalization

```
1:  procedure ProcessWindow(start, end)
2:      packets ← all (t, l) in packet_buffer where start ≤ t < end
3:      num_bins ← window_size / bin_width
4:      bins ← array of zeros of length num_bins
5:      for each (t, l) in packets do
6:          idx ← ⌊(t − start)/bin_width⌋
7:          if 0 ≤ idx < num_bins then
8:              bins[idx] ← bins[idx] + l
9:          end if
10:     end for
11:     norm_bins ← [ ]
12:     for each value b in bins do
13:         update running bins global_mean and global_std using b
14:         if global std > 0 then
15:             z ← (b - global_mean)/global_std
16:         else
17:             z ← 0
18:         end if
19:         append(norm_bins, z)
20:         UpdateWelford(z, num_bins)
21:     end for
22:     RS ← WelfordRS
23:     DDoSDecision(RS, end)
24: end procedure
```

Algorithm 3 UpdateWelford: Sliding window mean and variance

```
1:  procedure UpdateWelford(x_n, maxlen)
2:      if length(buffer) = maxlen then
3:          expired ← pop_left(buffer)
4:          n ← n − 1
5:          if n > 0 then
6:              δ_1 ← expired − x̄_n
7:              x̄_n ← x̄_n − δ_1/n
8:              δ_2 ← expired − x̄_n
9:              M_{2,n} ← M_{2,n} − (δ_1 · δ_2)
10:         end if
11:     end if
12:     append(buffer, x_n)
13:     n ← n + 1
14:     δ_1 ← x_n − x̄_n
15:     x̄_n ← x̄_n + δ_1/n
16:     δ_2 ← x_n − x̄_n
17:     M_{2,n} ← M_{2,n} + δ_1 · δ_2
18: end procedure
```

Algorithm 4 WelfordRS: R/S estimation

procedure WELFORDRS
  if $n \leq 1$ **then**
    return 0
  end if
  $S \leftarrow \sqrt{M_{2,n}/(n-1)}$ ▷ Compute standard deviation
  if $S = 0$ **then**
    return 0
  end if
  $Y_k \leftarrow x_k - \bar{x}_n \quad \forall k = 1, \ldots, n$
  $Z_k \leftarrow \sum_{t=1}^{k} Y_t \quad \forall k = 1, \ldots, n$
  $R \leftarrow \max_{1 \leq k \leq n} Z_k - \min_{1 \leq k \leq n} Z_k$ ▷ Compute range
  if $R = 0$ **then**
    return 0
  end if
  return $\log(R/\mathrm{S}) \ / \ \log(\mathrm{n})$
end procedure

Algorithm 5 DDoSDecision: Threshold-based DDoS detection

Require: Current $\log(R/S)/\log(n)$ value RS_t, history of normal window $\mathcal{RS}^N$, history of normal window variations $\Delta\mathcal{RS}^N$
Ensure: Detection flag (Normal or DDoS)
Compute $\Delta RS_t \leftarrow |RS_t - RS_{t-1}|$ (or 0 if unavailable)
if $|\mathcal{RS}^N| \geq 20$ **then**
  $\mu_{RS} \leftarrow \mathrm{mean}(\mathcal{RS}^N{}_{-20})$
  $\sigma_{RS} \leftarrow \mathrm{std}(\mathcal{RS}^N{}_{-20})$
  $RS_{\text{upper}} \leftarrow \mu_{RS} + 2.5\sigma_{RS}$
  $RS_{\text{lower}} \leftarrow \mu_{RS} - 2.5\sigma_{RS}$
else ▷ Initialization thresholds
  $RS_{\text{upper}} \leftarrow 0.67, \quad RS_{\text{lower}} \leftarrow 0.40$
end if
if $|\Delta\mathcal{RS}^N| \geq 2$ **then**
  $\mu_{\Delta RS} \leftarrow \mathrm{mean}(\Delta\mathcal{RS}^N{}_{-2})$
  $\sigma_{\Delta RS} \leftarrow \mathrm{std}(\Delta\mathcal{RS}^N{}_{-2})$
  $\Delta RS_{\text{th}} \leftarrow \mu_{\Delta RS} + 2\,\sigma_{\Delta RS}$
else
  $\Delta RS_{\text{th}} \leftarrow 0.15$
end if
if $(RS_t > RS_{\text{upper}} \vee RS_t < RS_{\text{lower}}) \ \wedge \ \Delta RS_t > \Delta RS_{\text{th}}$ **then**
  flag $\leftarrow$ DDoS
else
  flag $\leftarrow$ Normal
  Append RS_t to the history $\mathcal{RS}^N$
  Append ΔRS_t to the history $\Delta\mathcal{RS}^N$
end if

Algorithm 2 presents the PROCESSWINDOW procedure. The procedure starts by extracting packets that arrive within the window time from the buffer (Line 2). The window time is divided into equal-sized time bins (Lines 3–4). Then the lengths of the packets in each bin are summed to form a traffic series (Lines 5–10). This series is then normalized using a running mean and standard deviation to remove scale effects (Lines 12–19). Specifically, the running mean and standard deviation of the bin values are computed using Welford's online algorithm and applied to normalize the bins. Each bin value b is normalized using a z-score defined as

$$z = \frac{b - \mu}{\sigma}, \tag{17}$$

where μ and σ denote the running mean and standard deviation of the bin values, respectively. The normalized bin values are used to update Welford-based statistics. Note that the z-score normalization only removes differences in magnitude between bin values to standardize the scale of the traffic series. Therefore, it does not alter the temporal ordering or traffic characteristics. After that, the WELFORDRS procedure is called to compute R/S for the current window (Lines 20–22). This value is passed to the DDoS decision module to determine whether an attack is occurring (Line 23).

Algorithm 3 updates the mean and variance using a sliding window based on Welford's method. If the buffer is full, the algorithm removes the oldest value and decrements the sample count (Lines 2–4). Then, it performs a backward update, in which the mean and variance accumulators are updated to subtract the contribution of the removed value (Lines 6–9). Next, the new value is added to the buffer and the sample count is incremented (Lines 12–13). The algorithm then performs a forward update by computing the difference between the new and current mean values (Line 14) and use it to update the mean and variance accumulators (Lines 15–17). Through forward and backward updates, we incrementally update the mean and variance of the window without recomputing the statistics from scratch.

Algorithm 4 presents the WELFORDRS procedure. The procedure estimates the R/S using statistics maintained by Welford's method. First, the algorithm checks whether a sufficient number of samples is available. If fewer than two values are provided, the function immediately returns 0 (Lines 2–4). It then computes the standard deviation from the accumulated variance (Line 5) and performs an early exit if the deviation is zero (Lines 6–8). Next, the data in the sliding window are centered by subtracting the mean from each sample (Line 9), and a cumulative sum of these values is computed (Line 10). From this cumulative series, the range is computed as the difference between the maximum and minimum values (Line 11). If the computed range is zero, the procedure returns zero (Lines 12–14). Finally, the resulting estimate is obtained as the logarithm of the Rescaled Range divided by the window length (Line 15).

Algorithm 5 performs DDoS detection using a threshold-based decision mechanism and proceeds as follows. First, the algorithm computes the absolute difference between the current Rescaled Range value and the previous window to capture short-term changes in traffic behavior (Line 1). It then derives dynamic (i.e., adaptive) upper and lower thresholds for the R/S value using the mean and standard deviation of the most recent 20 windows classified as normal when sufficient history is available (Lines 2–6); otherwise, temporary thresholds are applied (Line 8). The values 0.4 and 0.67 are selected as temporary initial lower and upper thresholds, respectively. These values are used only until sufficient R/S values have been collected to compute the dynamic thresholds. Once sufficient historical data is available, these initial temporary threshold values are replaced with the dynamic thresholds computed using the most recent 20 windows. Based on our experiments, this choice provides the best balance between False Positives and detection delay.

Next, a dynamic difference threshold for the R/S variation is computed based on the most recent variation history (Lines 10–13), with a fallback constant of value 0.15 used when this history is insufficient (Lines 14–15). The fallback threshold value of 0.15 was selected based on experimental analysis of the difference in R/S variation between normal and attack traffic. This value is used only during the initialization phase. The algorithm then evaluates the current window R/S value against the computed thresholds. If the current R/S value violates the dynamic upper or lower bounds and the corresponding R/S variation exceeds its threshold, the window is labeled as a DDoS attack (Lines 17–19). Otherwise, the window is classified as normal traffic (Line 20). The proposed thresholding mechanisms are dynamic and adapt to changing network traffic conditions. To prevent

contaminating the adaptive thresholds by attack window R/S values, only windows classified as normal are used to update the $\mathcal{RS}^N$ and $\Delta\mathcal{RS}^N$ histories used for threshold computation (Lines 20–22). This prevents sustained attack windows from biasing the adaptive thresholds. Moreover, the proposed detection strategy is robust, as it requires two conditions to be satisfied to classify traffic as an attack, thereby reducing misclassifications.

5. Experimental Evaluation

All experiments were conducted on a virtual machine running Ubuntu 20.04. The virtual machine was hosted on a system equipped with a four-core processor operating at a base frequency of 1.80 GHz. We simulated the SDN environment using the Mininet emulator [53] and Ryu SDN controller [54]. Figure 3 shows the emulated Mininet topology, consisting of four switches and eight hosts. Both the controller and switches use the OpenFlow v1.3 protocol in the simulation. We used the Scapy tool to generate both normal and attack TCP and UDP traffic. The duration of the DDoS attack traffic was set to 10 s. In this experiment, two hosts, `h7` and `h8`, served as the attack sources for sending DDoS traffic. Traffic packets were captured in real time using Scapy sniffing, which was deployed on the Ryu application. For each packet, the arrival timestamp, protocol information, and packet length were saved.

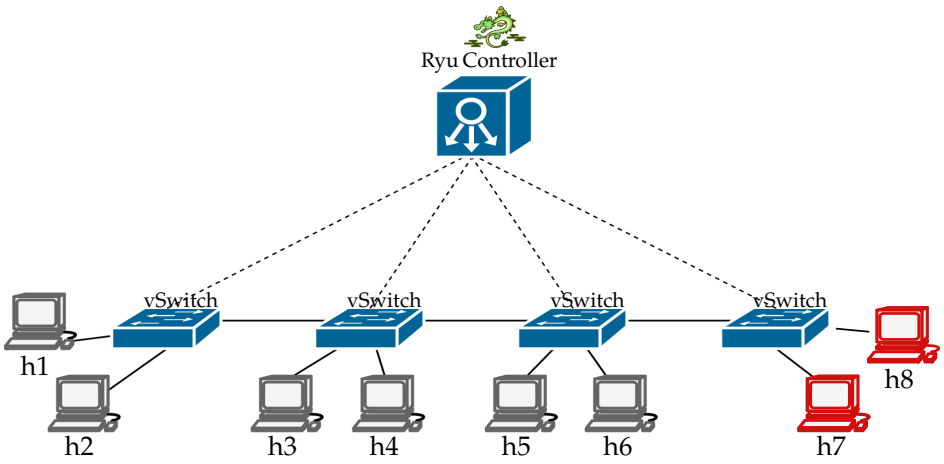

Figure 3. Mininet simulated topology.

We implemented our proposed sliding window R/S scheme using Python 3. For a comprehensive evaluation, we performed two complementary comparisons. First, we compared the proposed scheme with the Hurst exponent implemented in the `hurst` Python v3 library available on GitHub [55]. The `hurst` library estimates the Hurst index using multiple time scales followed by curve fitting and linear regression as described in Equations (1)–(9) previously. This comparison evaluates the detection performance and computational efficiency of the sliding window R/S method compared to the Hurst exponent, which involves multi-scale computation.

Second, we evaluated the use of Welford for R/S computation against three different R/S variants, namely, Traditional R/S, Simplified R/S, and NumPy R/S. This comparison evaluated the computation accuracy and computation time. The Traditional and Simplified variants were adopted from the `hurst` library [55]. The NumPy-based variant was implemented using `NumPy` vectorized operations. `NumPy` is an open-source, powerful Python library that enables efficient processing and fast computations of arrays and numerical data operations [56]. Its vectorized operations significantly accelerate computations on large time series. To make the comparison fair, all methods were compared using the same sliding window configurations as shown in Table 1.

Table 1. Sliding window configuration parameters.

Parameter	Value
Window size	2000 ms
Step size	500 ms
Bin width	20 ms
Number of bins	100
Overlap ratio	75%

6. Results and Discussion

In this section, we evaluate the effectiveness of the proposed sliding window R/S method by comparing it against the Hurst index and other R/S variants, namely, Traditional R/S, Simplified R/S, and NumPy-based R/S.

We aim to evaluate the efficiency, robustness, and accuracy of the proposed scheme in detecting DDoS attacks in OpenFlow-based SDN. The Traditional R/S variant follows the classical Rescaled Range (R/S) formulation as implemented in the Hurst Python 3 library available on GitHub [55]. The Simplified R/S, also taken from [55], offers a more basic calculation for a quick estimation of the Rescaled Range. It achieves this by omitting the cumulative deviation step; the range is computed directly from the observed values, whereas the standard deviation is still computed using the same steps as in the Traditional method. Although this variant significantly improves computational performance, it produces less accurate results than the full R/S analysis. The NumPy-based variant R/S utilizes the NumPy [56] library for range computation (i.e., np.ptp) to optimize the computations of the Traditional variant, but preserves all its mathematical steps and formulation.

The proposed Welford-based R/S scheme differs from the existing R/S variants by incrementally updating the mean and standard deviation as the window advances. In addition, it includes a backward update mechanism that removes the contribution of the expired sample from the window before adding the new sample via the forward update. The combination of forward and backward updates eliminates the need to recompute the statistics from scratch for each window. Hence, our scheme preserves the R/S formulation while substantially reducing the computational time and cost.

6.1. Comparing Proposed R/S to Hurst Index

Figure 4 presents the real-time comparison between the Hurst index and the proposed R/S method under normal and DDoS attack traffic. The Hurst-index computation follows the implementation provided by the Hurst library [55]. During the normal traffic phase (0–32 s), we observe that both methods fluctuate around 0.5, indicating that the traffic exhibits random, non-self-similar behavior. At the start of the DDoS attack (i.e., the shaded region), both methods exhibit a significant increase in their values, indicating the emergence of self-similar traffic introduced by the attack. It is clear that the proposed R/S exhibits a trend behavior similar to the Hurst index; whenever the Hurst index increases, R/S increases, and vice versa. This observation supports our claim that an exact estimate of the Hurst index is not necessary as long as its general behavior is captured; therefore, the proposed scheme can be effectively used.

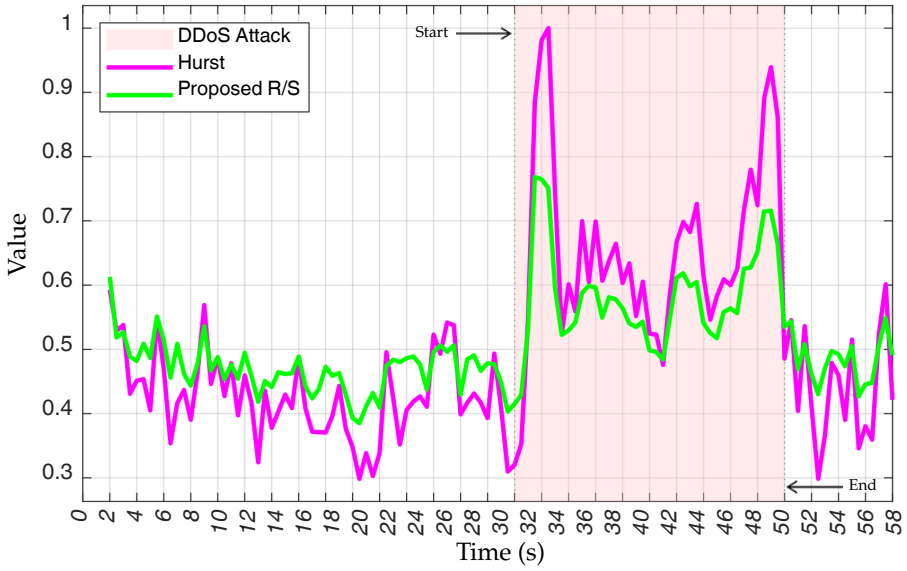

Figure 4. Real-time comparison of Hurst and proposed R/S methods under network traffic.

Figure 5 shows the distribution of the Hurst index and the proposed R/S values under normal and attack traffic. Under normal traffic, both methods have approximately the same values with a median around 0.5. However, under attack, both methods exhibit a noticeable increase in their values due to self-similar behavior. We observe that the Hurst-index values are more spread and variable than the proposed R/S, whereas the R/S values are more bounded and consistent. From these results, it is clear that the proposed R/S can effectively distinguish normal and attack traffic.

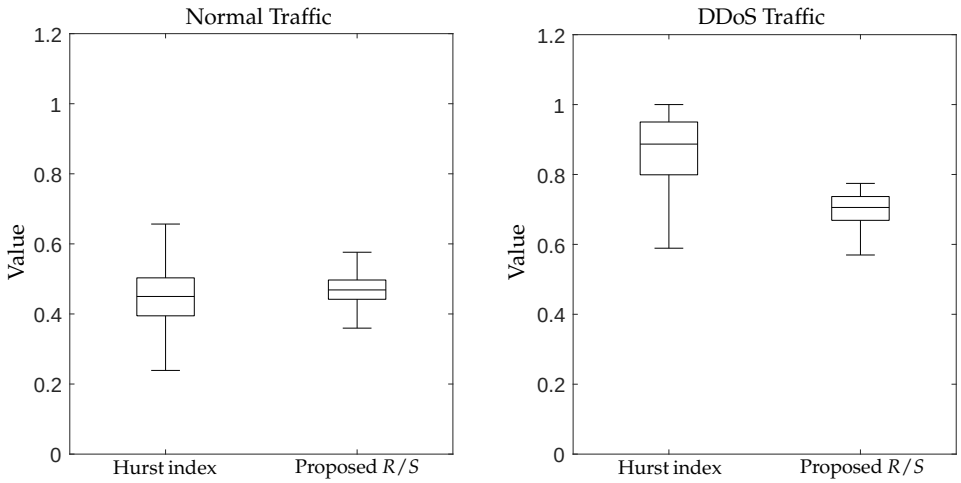

Figure 5. Distribution of values under normal and DDoS attack traffic for Hurst and proposed R/S.

The computation time per window of the Hurst index and the proposed R/S is shown in Figure 6. We can see that the Hurst index requires significantly longer computation time than the proposed R/S, with a median of 1.2 ms, due to the multi-scaling and curve fitting performed by the Hurst index. On the other hand, the proposed R/S outperforms Hurst index, achieving a lower computational time with a median of less than 0.04 ms. On average, the proposed R/S is approximately 32 times faster than the Hurst index as shown in Figure 7. This time reduction demonstrates the effectiveness and suitability of the proposed scheme in real-time networks.

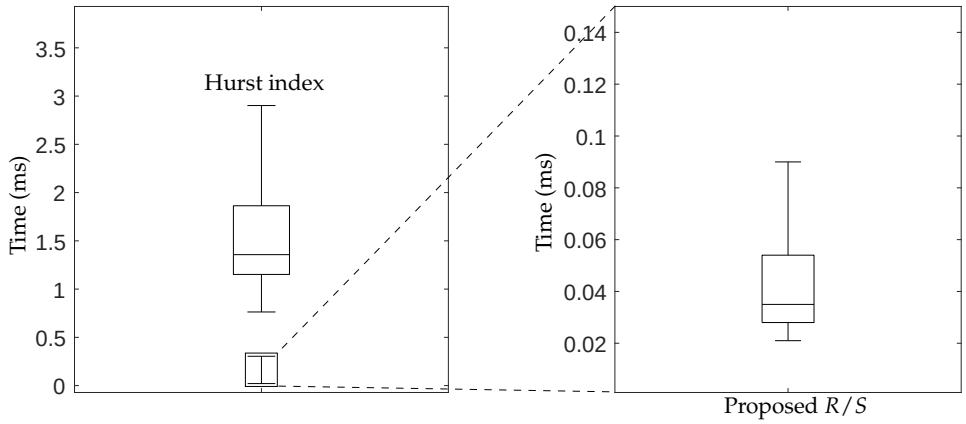

Figure 6. Box plot of computation times for Hurst index and proposed R/S.

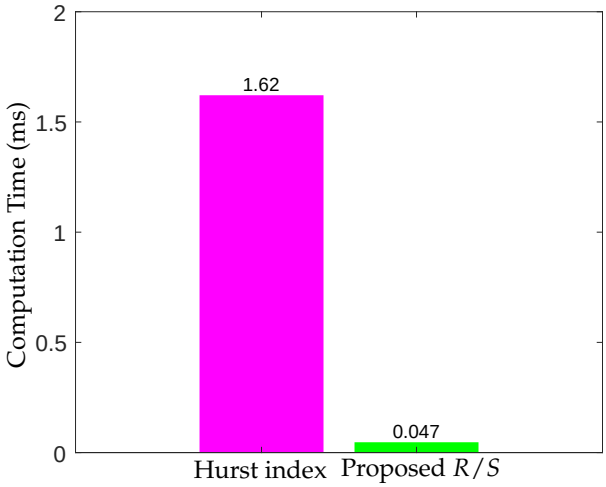

Figure 7. Average computation time of the Hurst and proposed R/S.

The comparison between the proposed R/S metric and the Hurst index confirms our claim that an exact estimation of the Hurst index is not necessary. Alternatively, an effective detection metric that follows the same temporal pattern as the Hurst index and responds reliably to changes in self-similarity is sufficient for detecting DDoS attacks.

6.2. Comparing the Proposed R/S to Other R/S Variants

In our evaluation, we compare our proposed Welford-based R/S with three publicly available implementations, namely, Traditional R/S, NumPy R/S, and Simplified R/S. This experiment demonstrates both the accuracy and efficiency of our algorithm in detecting DDoS attacks within a stream of normal traffic.

Figure 8 shows the box plot of the R/S values under normal and attack traffic for the different approaches. The box plot values are generated using approximately 3000 sample values obtained from multiple simulations. It is clear that the proposed scheme produces R/S values identical to those of the Traditional and NumPy methods. For normal traffic, the Traditional, NumPy, and proposed methods produced values around 0.45–0.5. Under attack conditions, the Traditional, NumPy, and proposed methods produced higher values around $\backsim$0.7, consistent with the expected emergence of self-similar traffic. In contrast, the Simplified R/S demonstrates noticeable deviations from other methods in both normal and attack traffic, resulting in reduced accuracy.

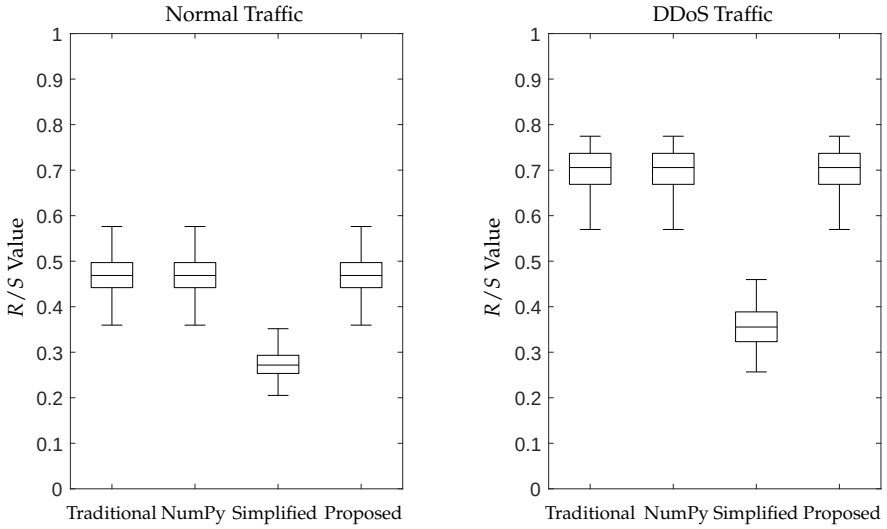

Figure 8. R/S values under normal and DDoS traffic.

The variations in R/S values produced by the four methods during normal and consecutive DDoS attacks in real time is illustrated in Figure 9. During normal traffic, R/S values fluctuate between 0.45 and 0.55 for all methods except the Simplified method. We observe that at the start and end of a DDoS attack, the R/S values of all methods, except the Simplified method, increase sharply to 0.75, capturing the presence of self-similar traffic. As mentioned previously, the Simplified method produces underestimated values for both normal and DDoS traffic and fails to capture changes in self-similarity. Both Figures 8 and 9 demonstrate that our optimized method achieves the same accuracy as the Traditional and NumPy methods.

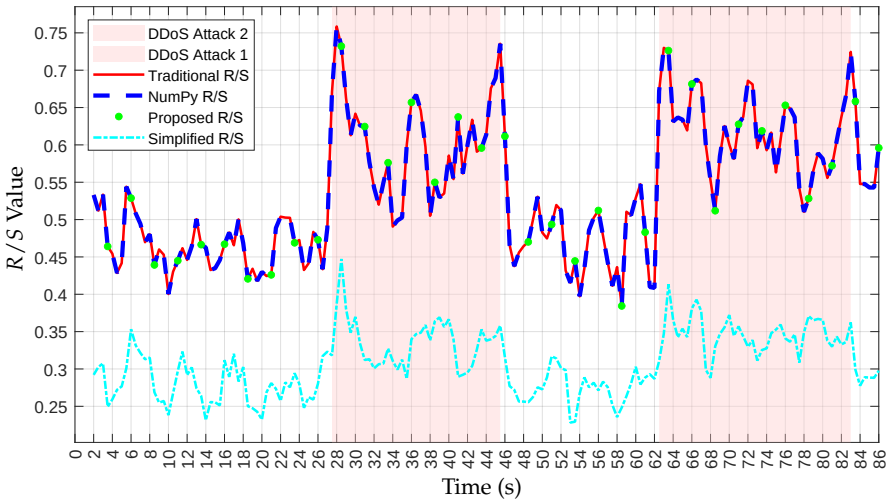

Figure 9. Real-time R/S values under two consecutive DDoS attacks.

We also simulated a more challenging scenario that includes high-variance but non-self-similar traffic (i.e., flash crowds, which consist of a large number of legitimate users attempting to access the same content simultaneously and are typically triggered by popular events like breaking news, etc. [57]) followed by a self-similar DDoS attack to test the robustness of our scheme as shown in Figure 10. We can see that during the flash crowd phase (45–55 s), the increase in traffic does not increase the R/S value, which is desirable because it is not self-similar, unlike DDoS traffic. The results highlight our scheme's ability to differentiate between flash crowds and DDoS traffic.

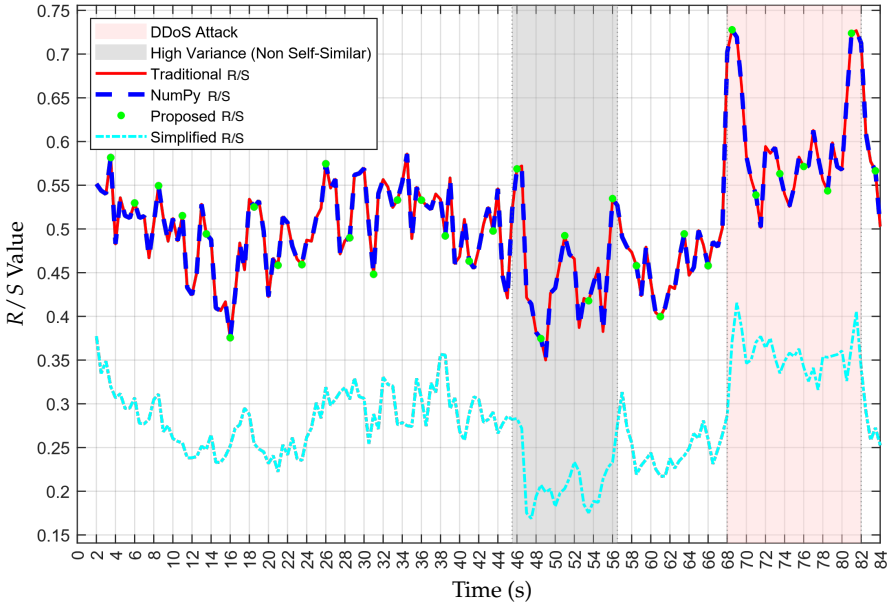

Figure 10. Real-time R/S values during high-variance normal traffic (non-self-similar) followed by a DDoS attack.

Moreover, we compare our proposed scheme to the other approaches in terms of computational time and speedup. We measured the computation time of the R/S value for each method and averaged the results across approximately 3000 values.

Figure 11 shows the box plot of the computational time of the R/S value per window of the four R/S methods. The Traditional method has the highest and most variable computation times, with a median around 0.25 ms and a maximum value of 0.5 ms. The NumPy and Simplified methods have noticeably improved computation time, with medians of 0.12 ms and 0.07 ms, respectively. However, our proposed scheme achieves the least variability and the lowest computational time with a median below 0.05 ms. The average computation time of our scheme is 0.046 ms, as shown in Figure 12. We additionally compared the speedup percentage of NumPy, Simplified, and our proposed Welford relative to the Traditional R/S. Although NumPy and Simplified methods achieve an observable speedup, the proposed scheme demonstrates superior performance, achieving the best speedup gain of 85.3% as illustrated in Figure 13. These results confirm the efficiency of the Welford online algorithm for computing R/S.

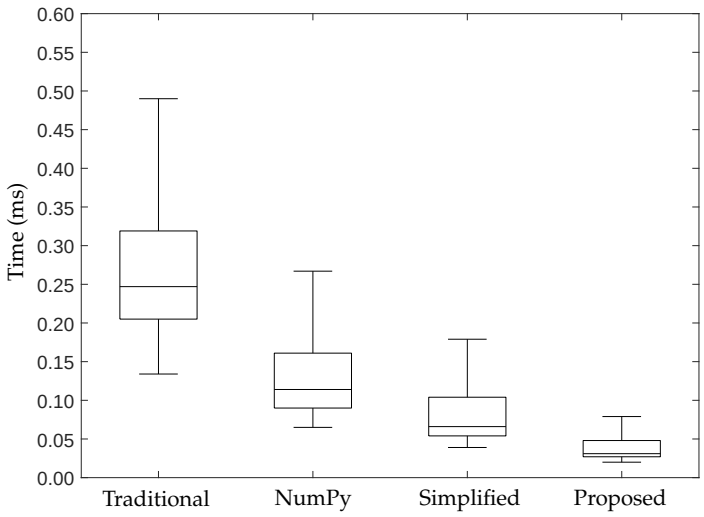

Figure 11. Box plot of computation times across R/S methods.

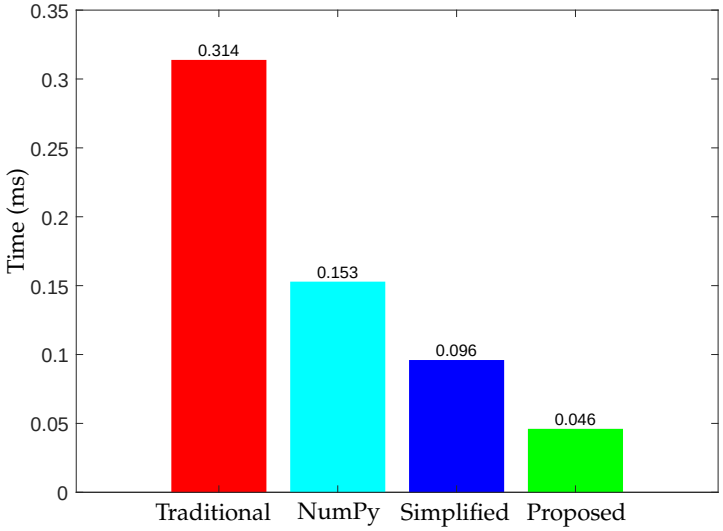

Figure 12. Average computation time of R/S methods.

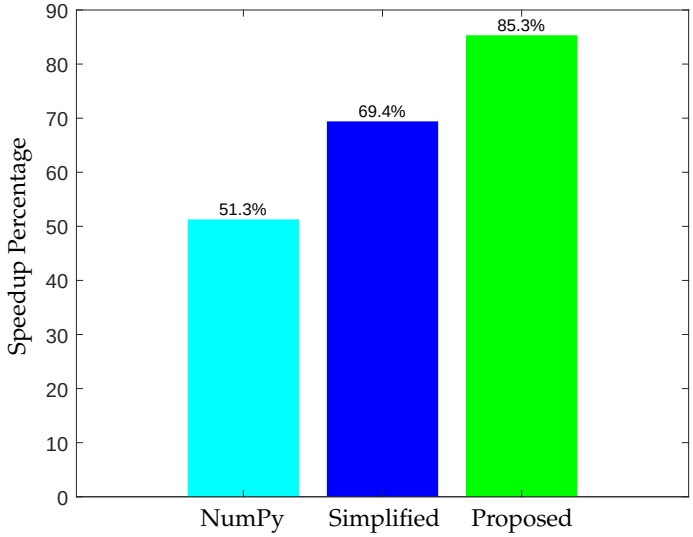

Figure 13. Speedup relative to Traditional R/S.

6.3. Detection Performance

For a comprehensive performance evaluation, we compare the proposed scheme against the Hurst index and other R/S variants using six performance metrics: False Positive Rate (FPR), False Negative Rate (FNR), accuracy, recall, precision, and F1-score. A True Positive (TP) corresponds to an attack window correctly classified as DDoS. A False Negative (FN) corresponds to an attack window incorrectly classified as normal. A True Negative (TN) represents a normal window correctly classified as normal, whereas a False Positive (FP) represents a normal window incorrectly classified as an attack. Precision measures the proportion of detected attack windows that are actual attacks; recall measures the detector's ability to detect actual attack windows. The F1-score is the harmonic mean of precision and recall [58]. The metrics are computed as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}; \tag{18}$$

$$\text{Precision} = \frac{TP}{TP + FP}; \tag{19}$$

$$\text{Recall} = \frac{TP}{TP + FN}; \tag{20}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}; \tag{21}$$

$$\text{FPR} = \frac{FP}{FP + TN}; \tag{22}$$

$$\text{FNR} = \frac{FN}{TP + FN} = 1 - \text{Recall}. \tag{23}$$

Tables 2 and 3 summarize the average detection performance of all methods over the TCP and UDP flooding attacks, respectively. It can be observed that, for both attacks, the Traditional, NumPy, and proposed R/S implementations (shown in bold) achieve approximately identical detection performance. In contrast, the Simplified R/S exhibits the weakest performance, achieving the lowest recall and F1-score and the highest FPR.

Under the TCP flooding scenario, the proposed scheme achieves 99.43% accuracy, 97.53% recall, and an FPR of 0.5%. Although the Hurst index achieves the lowest FPR (0.27%), it has lower recall and F1-score (86.23% and 86.59%, respectively), resulting in a higher FNR. Therefore, the proposed scheme provides a better balance between attack detection and false alarm minimization.

Table 2. Average detection performance under the TCP flooding attack.

Method	Accuracy	Precision	Recall	F1-Score	FPR	FNR
Hurst	0.9901	0.9235	0.8623	0.8659	0.0027	0.1377
NumPy R/S	0.9943	0.9179	0.9753	0.9346	0.0050	0.0247
Traditional R/S	0.9943	0.9179	0.9753	0.9346	0.0050	0.0247
Simplified R/S	0.9784	0.8726	0.8136	0.8194	0.0119	0.1864
Proposed *R/S*	**0.9943**	**0.9179**	**0.9753**	**0.9346**	**0.0050**	**0.0247**

Table 3. Average detection performance under the UDP flooding attack.

Method	Accuracy	Precision	Recall	F1-Score	FPR	FNR
Hurst	0.9918	0.8000	0.9130	0.8528	0.0045	0.0870
NumPy R/S	0.9920	0.7931	0.9500	0.8696	0.0054	0.0500
Traditional R/S	0.9920	0.7931	0.9500	0.8696	0.0054	0.0500
Simplified R/S	0.9817	0.5500	0.5789	0.5641	0.0093	0.4211
Proposed *R/S*	**0.9945**	**0.9091**	**0.9677**	**0.9375**	**0.0049**	**0.0323**

For the UDP flooding scenario, the proposed scheme and Hurst index achieved detection performance similar to the TCP flooding. The proposed scheme outperformed all other methods, achieving 99.45% accuracy, 96.77% recall, a 93.75% F1-score, and a 0.49% FPR. Also, it achieved the lowest FNR of 3.23%, indicating that only a small number of attack windows were missed. The performance of the NumPy and Traditional implementations has degraded slightly when compared to the TCP flooding results. However, the Simplified implementation showed the lowest performance, with its F1-score decreasing from 81.94% in TCP to 56.41% in UDP, because it trades R/S value accuracy for computational efficiency. This resulted in a high FNR value, indicating that it misses more attack windows than other methods. The results demonstrate that the proposed scheme generalizes well across different transport-layer flooding attacks.

The computation time reported in the previous subsections measures the execution time of all methods, but does not represent the actual detection delay. The detection delay

is the elapsed time from the start of the attack until the first DDoS flag, including the effect of sliding window accumulation. Table 4 summarizes the detection delay statistics over multiple DDoS scenarios. The proposed R/S implementation achieved the lowest mean detection delay, approximately 306.651 ms, followed by the NumPy and Traditional R/S implementations, which achieved close values of 306.679 ms and 306.801 ms, respectively. The three methods also exhibit similar median detection delays and comparable variability. This indicates that these methods consistently flag the current window as a DDoS attack. In contrast, the Simplified R/S and Hurst index achieved higher mean detection delay, 584.663 ms and 474.242 ms, respectively. Analyzing the experimental logs showed that these implementations occasionally classified the window as DDoS in subsequent sliding windows rather than the current one, introducing an additional delay of approximately one window step, ≈0.5 s. Consequently, the overall detection latency is affected by the window size and step size.

The overall results demonstrate the effectiveness of the proposed scheme compared to the Hurst index and other R/S variants. The results confirm that computing the Hurst index is not mandatory, because our aim is to have an efficient and reliable indicator that tracks changes in traffic self-similarity. The proposed scheme uses a fixed-size sliding window based on the R/S statistic to capture self-similarity changes in network traffic during a DDoS attack in real time while maintaining low computational complexity.

Table 4. Detection delay statistics.

Method	Mean ± Std. Dev. (ms)	Median (ms)
Hurst	474.243 ± 186.276	543.552
NumPy R/S	306.679 ± 199.967	284.398
Traditional R/S	306.801 ± 199.986	284.498
Simplified R/S	584.663 ± 208.554	586.603
Proposed *R/S*	**306.651 ± 199.956**	**284.378**

The experimental results show that the proposed scheme demonstrates superior detection performance for both TCP and UDP flooding attacks while maintaining low FPR and FNR. In addition, it can differentiate between a flash crowd and a real DDoS attack. Compared with the Hurst index, it achieves a comparable detection performance while reducing the FNR. The use of Welford's online algorithm for R/S calculation allows forward and backward incremental updates of the mean and variance without the need for recomputation each time the window slides. This reduces the computational time of the proposed scheme from 0.314 ms for the Traditional R/S method to 0.048 ms while maintaining the same accuracy as the Traditional and NumPy R/S implementations. On the other hand, the Simplified R/S is computationally efficient; however, its lower recall and F1-score make it less reliable for DDoS detection.

The detection delay analysis further demonstrates the effectiveness of the fixed-size sliding window in the R/S approach. The proposed Welford-based scheme achieves the lowest median detection delay of 284.378 ms, whereas the Hurst index, which relies on multi-scale computation, has a higher median detection delay of 543.552 ms. Consequently, the proposed scheme combines low computational overhead, high detection accuracy, and low detection delay, making it suitable for real-time DDoS detection in SDN. The proposed scheme is not limited to a specific SDN topology or specific network size, as it relies on the statistical characteristics of traffic passing through the network. Therefore, it can be deployed in various SDN environments, including enterprise, cloud, and data center networks. However, the window and step sizes may vary depending on traffic

characteristics and conditions. These parameters should be selected to achieve the best balance between detection accuracy and efficiency trade-off.

7. Conclusions

Self-similarity is an effective indicator of the presence of a DDoS attack in SDN; however, current methods (i.e., the Hurst index) are computationally inefficient. This paper proposes a lightweight, real-time DDoS detection scheme based on a sliding overlapping window of R/S values. Moreover, we optimize the R/S calculation using the Welford online algorithm that updates the mean and variance incrementally through forward and backward updates. Experiments were conducted in a Mininet-based environment managed by a Ryu controller, simulating both normal and TCP and UDP flooding DDoS attack traffic. The results demonstrate that the sliding overlapping window is sufficient for tracking self-similarity changes in traffic rather than the computationally expensive Hurst index being required. The proposed scheme consistently achieved high detection performance against both TCP and UDP attacks, with high accuracy, recall, and F1-score, and low FPR and FNR. Our scheme is 32 times faster than calculating the Hurst index, with comparable detection performance. Moreover, our scheme was compared to three R/S variants, namely, Traditional, Simplified, and NumPy. The proposed scheme achieves comparable accuracy to the Traditional R/S and NumPy R/S implementations while achieving a significant speedup over them. The speedup percentage of our scheme compared to the Traditional approach is 85.3%. Moreover, our scheme distinguishes between flash crowds and real DDoS attacks, achieving an FPR of only 0.5%. It also achieved the shortest median detection delay among all evaluated methods. In summary, the proposed scheme provides the best balance between computational efficiency, detection accuracy, and detection delay. The findings of this work present a promising scheme to improve SDN security against DDoS attacks, enabling real-time detection.

For future work, we plan to compare our scheme against machine learning techniques and alternative statistical methods to further validate its effectiveness. We aim to further optimize the calculation of the self-similarity metric to speed up detection. Finally, we aim to evaluate our scheme across different attack types, including low-rate attacks, and validate its performance using publicly available datasets.

Author Contributions: Conceptualization, M.K.A., G.A., and H.A.; methodology, M.K.A., G.A., H.A.; software, G.A. and H.A., D.H.A.D. and S.A.; validation, M.K.A., G.A., and H.A.; formal analysis, M.K.A., G.A., and H.A.; investigation, M.K.A. and D.H.A.D.; resources, M.K.A.; data curation, D.H.A.D.; writing original draft preparation, M.K.A., G.A., and H.A.; writing review and editing, M.K.A., D.H.A.D. and H.M.K.A.; visualization, M.K.A. and D.H.A.D.; supervision, M.K.A., and H.M.K.A.; project administration, M.K.A.; funding acquisition, M.K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Kuwait University Research, grant number EO03/24.

Data Availability Statement: The original data presented in the study are openly available at http://www.mohamadawad.com.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Anerousis, N.; Chemouil, P.; Lazar, A.A.; Mihai, N.; Weinstein, S.B. The Origin and Evolution of Open Programmable Networks and SDN. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1956–1971. [CrossRef]
2. Muzafar, S.; Jhanjhi, N.; Khan, N.A.; Ashfaq, F. DDoS Attack Detection Approaches in on Software Defined Network. In *2022 14th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*; IEEE: New York, NY, USA, 2022; pp. 1–5. [CrossRef]

3. Swami, R.; Dave, M.; Ranga, V. Detection and Analysis of TCP-SYN DDoS Attack in Software-Defined Networking. *Wirel. Pers. Commun.* **2021**, *118*, 2295–2317. [CrossRef]
4. P., S.; Kavin, B.P.; Srividhya, S.R.; V., R.; C., K.; Lai, W.C. Performance Evaluation of Stateful Firewall-Enabled SDN with Flow-Based Scheduling for Distributed Controllers. *Electronics* **2022**, *11*, 3000. [CrossRef]
5. Kazi, B.U.; Islam, M.K.; Siddiqui, M.M.H.; Jaseemuddin, M. A Survey on Software Defined Network-Enabled Edge Cloud Networks: Challenges and Future Research Directions. *Network* **2025**, *5*, 16. [CrossRef]
6. Ji, Z.; Cui, Y.; Guo, Y.; Shen, G.; Chen, Y.; Guo, C. Towards saturation attack detection in SDN: A multi-edge representation learning-based method. *J. King Saud Univ.-Comput. Inf. Sci.* **2025**, *37*, 138. [CrossRef]
7. Lara, A.; Kolasani, A.; Ramamurthy, B. Network Innovation using OpenFlow: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 493–512. [CrossRef]
8. Li, Z.; Xing, W.; Khamaiseh, S.; Xu, D. Detecting Saturation Attacks Based on Self-Similarity of OpenFlow Traffic. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 607–621. [CrossRef]
9. Tang, D.; Yan, Y.; Gao, C.; Liang, W.; Jin, W. LtRFT: Mitigate the Low-Rate Data Plane DDoS Attack with Learning-to-Rank Enabled Flow Tables. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3143–3157. [CrossRef]
10. Shahzad, N.; Mujtaba, G.; Elahi, M. Benefits, Security and Issues in Software Defined Networking (SDN). *NUST J. Eng. Sci.* **2015**, *8*, 38–43. [CrossRef]
11. Kaur, S.; Kumar, K.; Aggarwal, N. Analysis of DDoS Attacks in Software Defined Networking. In *2022 IEEE Delhi Section Conference (DELCON)*; IEEE: New York, NY, USA, 2022; pp. 1–6. [CrossRef]
12. Karnani, S.; Agrawal, N.; Kumar, R. A comprehensive survey on low-rate and high-rate DDoS defense approaches in SDN: Taxonomy, research challenges, and opportunities. *Multimed. Tools Appl.* **2024**, *83*, 35253–35306.
13. Li, M. Change trend of averaged Hurst parameter of traffic under DDOS flood attacks. *Comput. Secur.* **2006**, *25*, 213–220. [CrossRef]
14. Sheng, Z.; Qifei, Z.; Xuezeng, P.; Xuhui, Z. Detection of low-rate DDoS attack based on self-similarity. In *2010 Second International Workshop on Education Technology and Computer Science*; IEEE: New York, NY, USA, 2010; Volume 1, pp. 333–336.
15. Smiesko, J.; Segec, P.; Kontsek, M. Machine recognition of DDoS attacks using statistical parameters. *Mathematics* **2023**, *12*, 142. [CrossRef]
16. Hajtmanek, R.; Kontšek, M.; Smieško, J.; Uramová, J. One-parameter statistical methods to recognize DDoS attacks. *Symmetry* **2022**, *14*, 2388. [CrossRef]
17. Awad, M.K.; Alsholi, G.; Altabaa, H.; Abu Daqar, D.H.; Alshaher, S.; Alazemi, H.M. Self-similarity-based DDoS Detection for Software-defined Networks. In *2025 International Conference on Computing, Networking and Communications (ICNC)*; IEEE: New York, NY, USA, 2025; pp. 185–189. [CrossRef]
18. Kaur, G.; Saxena, V.; Gupta, J. Detection of TCP targeted high bandwidth attacks using self-similarity. *J. King Saud Univ.-Comput. Inf. Sci.* **2020**, *32*, 35–49. [CrossRef]
19. Mukherjee, S.; Sadhukhan, B.; Das, A.K.; Chaudhuri, A. Hurst exponent estimation using neural network. *Int. J. Comput. Sci. Eng.* **2023**, *26*, 157. [CrossRef]
20. Welford, B.P. Note on a method for calculating corrected sums of squares and products. *Technometrics* **1962**, *4*, 419–420. [CrossRef]
21. Hirsi, A.; Alhartomi, M.A.; Audah, L.; Salh, A.; Sahar, N.M.; Ahmed, S.; Ansa, G.O.; Farah, A. Comprehensive Analysis of DDoS Anomaly Detection in Software-Defined Networks. *IEEE Access* **2025**, *13*, 23013–23071. [CrossRef]
22. Wang, H.; Li, Y. Overview of DDoS Attack Detection in Software-Defined Networks. *IEEE Access* **2024**, *12*, 38351–38381. [CrossRef]
23. Wabi, A.A.; Idris, I.; Olaniyi, O.M.; Ojeniyi, J.A. DDOS attack detection in SDN: Method of attacks, detection techniques, challenges and research gaps. *Comput. Secur.* **2024**, *139*, 103652. [CrossRef]
24. Khamaiseh, S.; Al-Alaj, A.; Adnan, M.; Alomari, H.W. The Robustness of Detecting Known and Unknown DDoS Saturation Attacks in SDN via the Integration of Supervised and Semi-Supervised Classifiers. *Future Internet* **2022**, *14*, 164. [CrossRef]
25. Ling, Y.; Yang, C.; Li, X.; Xie, M.; Ming, S.; Lu, J.; Tang, F. Real-time detection of DDoS attacks based on hurst index. In *2022 2nd International Conference on Networking Systems of AI (INSAI)*; IEEE: New York, NY, USA, 2022; pp. 42–45.
26. Mchina, J.P.; Mduma, N.; Sinde, R.S. Adaptive Decision-Level Intrusion Detection for Known and Zero-Day Attacks. *Network* **2026**, *6*, 23. [CrossRef]
27. Tanimu, J.A.; Bendiab, G.; Kanta, A.; Shiaeles, S. AI-Driven Threat Detection and Automated Incident Response for Enhancing Network Security. *Network* **2026**, *6*, 32. [CrossRef]
28. Ahuja, N.; Singal, G.; Mukhopadhyay, D.; Kumar, N. Automated DDOS attack detection in software defined networking. *J. Netw. Comput. Appl.* **2021**, *187*, 103108. [CrossRef]
29. Garba, U.H.; Toosi, A.N.; Pasha, M.F.; Khan, S. SDN-based detection and mitigation of DDoS attacks on smart homes. *Comput. Commun.* **2024**, *221*, 29–41. [CrossRef]

30. Srivastava, A.; Tiwari, S.; Kumar, D.; Garg, N. Finding of ddos attack in iot-based networks using ensemble technique. In *2024 International Conference on Intelligent Systems for Cybersecurity (ISCS)*; IEEE: New York, NY, USA, 2024; pp. 1–4.
31. Hnamte, V.; Najar, A.A.; Nhung-Nguyen, H.; Hussain, J.; Sugali, M.N. DDoS attack detection and mitigation using deep neural network in SDN environment. *Comput. Secur.* **2024**, *138*, 103661. [CrossRef]
32. Clinton, U.B.; Hoque, N.; Robindro Singh, K. Classification of DDoS attack traffic on SDN network environment using deep learning. *Cybersecurity* **2024**, *7*, 23. [CrossRef]
33. Aladaileh, M.A.; Anbar, M.; Hintaw, A.J.; Hasbullah, I.H.; Bahashwan, A.A.; Al-Amiedy, T.A.; Ibrahim, D.R. Effectiveness of an entropy-based approach for detecting low-and high-rate DDoS attacks against the SDN controller: Experimental analysis. *Appl. Sci.* **2023**, *13*, 775. [CrossRef]
34. Shirsath, V.A.; Chandane, M.M.; Lal, C.; Conti, M. SYNTROPY: TCP SYN DDoS attack detection for Software Defined Network based on Rényi entropy. *Comput. Netw.* **2024**, *244*, 110327. [CrossRef]
35. Swami, R.; Dave, M.; Ranga, V. IQR-based approach for DDoS detection and mitigation in SDN. *Def. Technol.* **2023**, *25*, 76–87. [CrossRef]
36. Bhayo, J.; Jafaq, R.; Ahmed, A.; Hameed, S.; Shah, S.A. A Time-Efficient Approach Toward DDoS Attack Detection in IoT Network Using SDN. *IEEE Internet Things J.* **2022**, *9*, 3612–3630. [CrossRef]
37. Huang, Z.; Huang, X.; Li, J.; Xue, K.; Sun, Q.; Lu, J. LLDM: Low-Latency DoS Attack Detection and Mitigation in SDN. In *2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR)*; IEEE: New York, NY, USA, 2022; pp. 169–174. [CrossRef]
38. Kalash, S.; Makarem, N.; Issa, L.; Tajeddine, A.; Abbas, N. Detection and Prevention of TCP DoS/DDoS Attacks in Software Defined Network. In *2023 IEEE 4th International Multidisciplinary Conference on Engineering Technology (IMCET)*; IEEE: New York, NY, USA, 2023; pp. 50–55. [CrossRef]
39. Maddu, J.S.; Tripathy, S.; Nayak, S.K. SDNGuard: An Extension in Software Defined Network to Defend DoS Attack. In *2019 IEEE Region 10 Symposium (TENSYMP)*; IEEE: New York, NY, USA, 2019; pp. 44–49. [CrossRef]
40. Bhuiyan, Z.A.; Islam, S.; Islam, M.M.; Ullah, A.B.M.A.; Naz, F.; Rahman, M.S. On the (in)Security of the Control Plane of SDN Architecture: A Survey. *IEEE Access* **2023**, *11*, 91550–91582. [CrossRef]
41. Kalambe, D.; Sharma, D.; Kadam, P.; Surati, S. A comprehensive plane-wise review of DDoS attacks in SDN: Leveraging detection and mitigation through machine learning and deep learning. *J. Netw. Comput. Appl.* **2025**, *235*, 104081. [CrossRef]
42. Ali, T.E.; Chong, Y.W.; Manickam, S. Machine learning techniques to detect a (DDoS) attack in (SDN): A systematic review. *Appl. Sci.* **2023**, *13*, 3183. [CrossRef]
43. Beran, J.; Feng, Y.; Ghosh, S.; Kulik, R. *Long-Memory Processes: Probabilistic Properties and Statistical Methods*; Springer: Berlin/Heidelberg, Germany, 2013. [CrossRef]
44. Mandelbrot, B.B.; Van Ness, J.W. Fractional Brownian motions, fractional noises and applications. *SIAM Rev.* **1968**, *10*, 422–437. [CrossRef]
45. Song, W.; Beshley, M.; Przystupa, K.; Beshley, H.; Kochan, O.; Pryslupskyi, A.; Pieniak, D.; Su, J. A software deep packet inspection system for network traffic analysis and anomaly detection. *Sensors* **2020**, *20*, 1637. [CrossRef] [PubMed]
46. Hurst, H.E. Long-Term Storage Capacity of Reservoirs. *Trans. Am. Soc. Civ. Eng.* **1951**, *116*, 770–799. [CrossRef]
47. Tang, D.; Feng, Y.; Zhang, S.; Qin, Z. FR-RED: Fractal residual based real-time detection of the LDoS attack. *IEEE Trans. Reliab.* **2020**, *70*, 1143–1157. [CrossRef]
48. Chan, T.F.; Golub, G.H.; LeVeque, R.J. Algorithms for computing the sample variance: Analysis and recommendations. *Am. Stat.* **1983**, *37*, 242–247. [CrossRef]
49. Ling, R.F. Comparison of several algorithms for computing sample means and variances. *J. Am. Stat. Assoc.* **1974**, *69*, 859–866. [CrossRef]
50. Efanov, A.A.; Ivliev, S.A.; Shagraev, A.G. Welford's algorithm for weighted statistics. In *2021 3rd International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE)*; IEEE: New York, NY, USA, 2021; pp. 1–5. [CrossRef]
51. Hanson, R.J. Stably updating mean and standard deviation of data. *Commun. ACM* **1975**, *18*, 57–58. [CrossRef]
52. West, D. Updating mean and variance estimates: An improved method. *Commun. ACM* **1979**, *22*, 532–535.
53. Lantz, B.; Heller, B.; McKeown, N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets-IX)*; ACM: New York, NY, USA, 2010; pp. 1–6. [CrossRef]
54. Ryu Project Team. Ryu SDN Framework. 2014. Available online: https://ryu-sdn.org/ (accessed on 15 August 2025).
55. Mottl, D. hurst: Hurst Exponent Evaluation and R/S-Analysis. 2022. Available online: https://github.com/Mottl/hurst (accessed on 15 June 2025).
56. van der Walt, S.; Colbert, S.C.; Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.* **2011**, *13*, 22–30. [CrossRef]

57. Jung, J.; Krishnamurthy, B.; Rabinovich, M. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the 11th International Conference on World Wide Web*; ACM: New York, NY, USA, 2002; pp. 293–304. [CrossRef]
58. Fu, Y.; Zhao, H.; Zhang, H.; Ma, J.; Liu, L. Fault detection and diagnosis of the AHU via combining a new hybrid monitoring strategy and an improved graph convolutional network. *Energy Build.* **2026**, *358*, 117177. [CrossRef]