electronics

Article

High Performance Network Intrusion Detection System Using Two-Stage LSTM and Incremental Created Hybrid Features

Jonghoo Han and Wooguil Pak *

Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, Republic of Korea
* Correspondence: wooguilpak@yu.ac.kr; Tel.: +82-53-810-3092

Abstract: Currently, most network intrusion detection systems (NIDSs) use information about an entire session to detect intrusion, which has the fatal disadvantage of delaying detection. To solve this problem, studies have been proposed to detect intrusions using only some packets belonging to the session but have limited effectiveness in increasing the detection performance compared to conventional methods. In addition, space complexity is high because all packets used for classification must be stored. Therefore, we propose a novel NIDS that requires low memory storage space and exhibits high detection performance without detection delay. The proposed method does not need to store packets for the current session and uses only some packets, as in conventional methods, but achieves very high detection performance. Through experiments, it was confirmed that the proposed NIDS uses only a small memory of 25.8% on average compared to existing NIDSs by minimizing memory consumption for feature creation, while its intrusion detection performance is equal to or higher than those of existing ones. As a result, this method is expected to significantly help increase network safety by overcoming the disadvantages of machine-learning-based NIDSs using existing sessions and packets.

Keywords: hybrid feature; network intrusion detection; session feature; packet feature; two-stage LSTM

Citation: Han, J.; Pak, W. High Performance Network Intrusion Detection System Using Two-Stage LSTM and Incremental Created Hybrid Features. *Electronics* **2023**, *12*, 956. https://doi.org/10.3390/electronics12040956

Academic Editors: Yu-an Tan, Qikun Zhang, Yuanzhang Li and Xiao Yu

Received: 19 January 2023
Revised: 13 February 2023
Accepted: 13 February 2023
Published: 15 February 2023

1. Introduction

Currently, a network intrusion detection system (NIDS) manages traffic by dividing it into logical groups called sessions. Therefore, it should collect the necessary information by creating state information for the current active session and monitoring traffic [1–5], implying that an NIDS with finite computing and memory resources has a limited number of simultaneous sessions that can be handled. Owing to the steadily increasing size of networks and the increasing amount of network traffic, the number of sessions that an NIDS has to handle is also increasing [6].

Accurately distinguishing between network intrusion and normal sessions requires much information about the session. To date, many NIDSs have stored the entire traffic belonging to the session, used it to analyze the total traffic sent and received after the session ended, extracted its statistical characteristics, and used them to distinguish intrusion from normal sessions [1–4]. However, collecting all traffic during a session's lifetime requires excessive storage space. Instead of collecting traffic, it can collect only a portion of the packet, such as the packet header, and create features for the session; however, using storage space in proportion to the number of packets remains a problem.

An NIDS using session characteristics requires a high storage space and also has the disadvantage of detecting intrusions only after the session is terminated. To solve these problems, a new type of NIDS that collects only session initial packets and uses them for detection has been proposed [5], where, instead of collecting traffic, it can collect only a portion of the packet, such as the packet header, and create features for the session. However, this approach still uses storage space in proportion to the number of packets.

To solve the problem of existing NIDSs, the proposed method directly inputs the received packet data to the classifier without collecting the packet data and stores the output through it. In addition, when the next session packet is received, the previously stored output and received new packet are input back to the classifier; therefore, partial classification is performed every time a packet is received. Further, whenever a new session packet is received, several state values for the session are updated, and a feature set of the machine-learning (ML) model is finally created using these values. In addition, instead of using all packets for each session, intrusion detection is performed before session termination because only some packets are used at the beginning of the session, as in the conventional method.

Therefore, this study makes the following contributions.

- 002D presents an NIDS model that progressively classifies using an ML model every time a packet is received.
- Suggests a hybrid feature creation method that uses packet and session data simultaneously.
- Proposes an NIDS capable of fast intrusion detection while maintaining high detection performance using partial packet data instead of entire session packets.

The remainder of this study is organized as follows. Section 2 briefly summarizes the related research work. Section 3 describes the proposed NIDS, while Section 4 compares and analyzes the performance of the proposed system with that of the latest competing work. Finally, Section 5 presents the conclusions of the study.

2. Existing Work

2.1. Session-Based NIDS

A machine learning-based NIDS (ML-NIDS) converts received traffic into features and classifies them to detect network intrusions. At this time, an ML-NIDS is classified according to how the received traffic is converted into features. Traffic is divided into sessions and monitored, and its nature (intrusion or normal traffic) is determined. Therefore, to convert traffic into features, it is divided into a specific session, and then features for the session are created using traffic belonging to the session [7–13]. A session is a concept that exists in the Transmission Control Protocol (TCP), but the User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) also extend the concept of TCP to define sessions. The method primarily used at this time classifies traffic with a key of five tuples: same source IP address, destination IP address, source port, destination port, and protocol. Packets with the same five tuples are considered to belong to the same session, but UDP does not have a packet indicating the end of the session. Therefore, if the packet inter-arrival time (IAT) of two adjacent packets exceeds a certain value (i.e., maximum IAT), the existing session is considered to be terminated. Subsequent packets are considered to belong to another session. In ICMP, a session can be defined as four tuples using the source IP address, destination IP address, type, and protocol fields, rather than five tuples. For this protocol, the end of the session is also defined using the IAT of the adjacent packets.

Thus, when defining the start and end of a session with the value of the IAT, it should be noted that even in TCP, a retransmitted packet may be received during session termination or a timeout may occur owing to packet loss [14]. Ultimately, the maximum IAT for a session in an NIDS should be set to a sufficiently large value to handle these situations. If the maximum IAT is set, packets whose IATs exceed this value are not processed as packets belonging to the existing session, even if they belong to the existing session. Instead, they are treated as packets belonging to a completely new session with the same five tuples. Therefore, the maximum IAT must be carefully assigned. It is assigned a value of 30 to 180 s, although it varies depending on the type of NIDS [15].

Although the maximum IAT is an important value for distinguishing between sessions, it can negatively affect the NIDS detection of intrusions because a delay equal to the maximum IAT necessarily occurs between the times when a session is terminated and when the NIDS determines that the session is terminated. Nevertheless, traffic is classified as a unit of session because NIDS requires tremendous hardware performance

to determine whether an intrusion has occurred for every individual packet received. A packet processing performance of up to 30 million packets per second must be supported to process 10 Gbps ethernet traffic without loss, implying that the NIDS must perform deep learning ML classification 30 million times per second, which is difficult to support even with dedicated hardware accelerators. Network traffic should be classified in any unit, i.e., single packet or session, and determining intrusion for each session is the most realistic implementation approach in current hardware technology.

2.2. Session Features

In this section, converting the traffic of each session into an ML input when the network traffic is divided into sessions is explained. The most common approach in existing studies is to extract multiple characteristics for a single session through statistical analysis of session traffic. This method extracts simple characteristics, such as the total data amount of session traffic and the total number of packets of session traffic, as well as those that require total traffic, such as the average value of the IAT or standard deviation. An example of a session feature is presented in Table 1 [9–13].

We call the session information obtained in this manner a session feature. An NIDS using the session feature must collect the entire traffic of each session until it is terminated. Therefore, an NIDS using session features collects intra-session traffic, analyzes session traffic after session termination, creates session features, and uses them to detect intrusions, as shown in Figure 1.

At this time, to detect session termination, the user must wait for the maximum IAT after the actual session termination. Therefore, if a network intrusion occurs, it is detected only after the maximum IAT has elapsed after the actual session termination [10–20]. Therefore, session-based NIDSs have the major disadvantage of long delays in detecting intrusions. Additionally, because the session feature is created using the total traffic per session, a considerable amount of traffic must be stored. Storage space may be saved by only storing packet headers, instead of storing raw packets or storing only some information necessary to create features. However, this process also requires storage space proportional to the number of packets belonging to the session. In addition, an NIDS using session features must decide in advance which characteristics of the session to use as features. These decisions are determined by the designer, with the performance of an NIDS significantly affected by the type of feature selected. Furthermore, when a new attack emerges, existing features may be insufficient to detect it. In this case, a new feature must be designed, posing a considerable burden on NIDS developers. Table 2 lists the strengths and weaknesses of session-based NIDS.

Table 1. ISCXIDS 2012 feature list showing 'act': active, 'avg': average, 'blk': block, 'bwd': backward, 'byts': bytes, 'cnt': count, 'CWR': congestion window reduced flag, 'dst': destination, 'ECE': explicit congestion notification for echo flag, 'fwd': forward, 'IAT': inter-arrival-time, 'init': initial, 'len': length, 'pkt': packet, 'PSH': push flag, 'RST': reset flag, 'seg': segment, 'src': source, 'std': standard deviation, 'tot': total, 'URG': urgent flag, 'var': variance, 'win': window.

No.	Name	No.	Name	No.	Name	No.	Name	No.	Name
1	Src IP	18	Bwd Pkt Len Max	35	Bwd IAT Std	52	SYN Flag Cnt	69	Subflow Fwd Pkts
2	Src Port	19	Bwd Pkt Len Min	36	Bwd IAT Max	53	RST Flag Cnt	70	Subflow Fwd Byts
3	Dst IP	20	Bwd Pkt Len Mean	37	Bwd IAT Min	54	PSH Flag Cnt	71	Subflow Bwd Pkts
4	Dst Port	21	Bwd Pkt Len Std	38	Fwd PSH Flags	55	ACK Flag Cnt	72	Subflow Bwd Byts
5	Protocol_HOPOPT	22	Flow Byts/s	39	Bwd PSH Flags	56	URG Flag Cnt	73	Init Fwd Win Byts
6	Protocol_TCP	23	Flow Pkts/s	40	Fwd URG Flags	57	CWE Flag Count	74	Init Bwd Win Byts
7	Protocol_UDP	24	Flow IAT Mean	41	Bwd URG Flags	58	ECE Flag Cnt	75	Fwd Act Data Pkts
8	Timestamp	25	Flow IAT Std	42	Fwd Header Len	59	Down/Up Ratio	76	Fwd Seg Size Min
9	Flow Duration	26	Flow IAT Max	43	Bwd Header Len	60	Pkt Size Avg	77	Active Mean
10	Tot Fwd Pkts	27	Flow IAT Min	44	Fwd Pkts/s	61	Fwd Seg Size Avg	78	Active Std
11	Tot Bwd Pkts	28	Fwd IAT Tot	45	Bwd Pkts/s	62	Bwd Seg Size Avg	79	Active Max
12	TotLen Fwd Pkts	29	Fwd IAT Mean	46	Pkt Len Min	63	Fwd Byts/b Avg	80	Active Min
13	TotLen Bwd Pkts	30	Fwd IAT Std	47	Pkt Len Max	64	Fwd Pkts/b Avg	81	Idle Mean
14	Fwd Pkt Len Max	31	Fwd IAT Max	48	Pkt Len Mean	65	Fwd Blk Rate Avg	82	Idle Std
15	Fwd Pkt Len Min	32	Fwd IAT Min	49	Pkt Len Std	66	Bwd Byts/b Avg	83	Idle Max
16	Fwd Pkt Len Mean	33	Bwd IAT Tot	50	Pkt Len Var	67	Bwd Pkts/b Avg	84	Idle Min
17	Fwd Pkt Len Std	34	Bwd IAT Mean	51	FIN Flag Cnt	68	Bwd Blk Rate Avg	85	Label

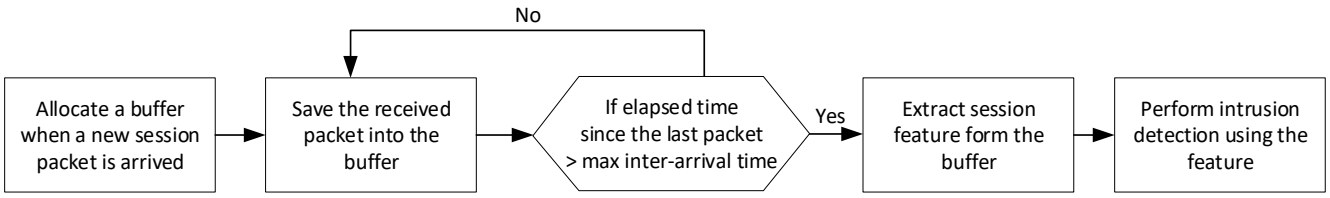


Figure 1. Typical procedure of the session-feature-based-NIDS to process the session.

Table 2. Advantages and disadvantages of session-feature-based NIDS.

Advantages	▪ Intrusion detection is performed at every session termination instead of every received packet, reducing computational overhead required for detection. ▪ Robust against loss of some packets.
Disadvantages	▪ Huge memory requirement to store received packets per session. ▪ The NIDS developer must select and design the feature type. ▪ When a new attack emerges, it is likely that a new feature type will have to be designed. ▪ Detection delay occurs due to waiting for the maximum inter-arrival time to recognize session termination. ▪ Vulnerable to NIDS detection bypass attacks through repetitive transmission of packets by an attacker to change important session feature values.

2.3. Packet Features

Although the session feature has excellent advantages, it also has many fatal disadvantages, necessitating studies for its improvement. Among them, the most effective alternative involves using the traffic belonging to the session as a feature [5]. Using raw packet data directly as a feature fundamentally avoids determining the type of session feature in advance. As the ML model automatically determines the most effective characteristics for existing attacks, it is sufficient to train the model again with the updated dataset when the session data for a new type of intrusion are added. However, because raw packet data are used as a feature, the packet itself must be stored, placing a heavy burden on the storage space per session. Thus, only part of the packet is stored, with only some of the session's initial packets used instead of all the packets. Let us call the features directly transformed from raw packet data packet features. The packet feature uses some session packets, but the actual detection performance is comparable to that of the existing session-feature-based NIDS. In addition, unlike the session feature, maximum IAT need not be used because there is no need to wait until the session ends. Therefore, the NIDS can rapidly detect an intrusion after it has occurred, which is the most important characteristic of the packet-feature-based method; thus, the packet-feature-based NIDS is advantageous in securing networks.

As mentioned earlier, session features and packet features have different characteristics because the methods of creating features are completely different. Considering session and packet features can complement each other's disadvantages, research on a new type of feature that further strengthens the advantages of the two features and improves the disadvantages is continuously required.

However, in the field of NIDS, there is no research to apply both features at the same time. In order to apply both feature types simultaneously, there are problems to be solved. First, the memory usage required for feature creation can increase significantly. Currently, even NIDS using only session features or packet features requires high memory usage, so it is technically very difficult to create both features at the same time. In addition to this, we need a machine learning model that can apply both features together. While all packets belonging to each session are required to create a session feature, only some packets at the

beginning of a session are used to create a packet feature. Therefore, a new NIDS design is needed to combine the advantages of both methods rather than simply using them. Table 3 lists the strengths and weaknesses of packet-based NIDS.

Table 3. Advantages and disadvantages of packet-feature-based NIDS.

Advantages	▪ When a new attack emerges, it can automatically create features to detect the new attack, eliminating the need for NIDS developer intervention. ▪ Intrusion detection is possible before session termination. ▪ When it handles long sessions, there is no detection delay due to the maximum inter-arrival time for end-of-session recognition. ▪ It supports deep packet inspection. ▪ Detection rate is very high.
Disadvantages	▪ High memory requirement to store some packet data per session. ▪ Patterns located in packet payloads that are not used as features cannot be detected.

3. Materials and Methods

3.1. Motivation

According to previous studies on NIDS, session and packet features have significantly different characteristics, which can considerably improve detection performance through synergistic effects if they can be used together. However, the amount of information to be stored and maintained increases excessively when both feature types are used simultaneously. In addition, the detection delay may intensify as the time required for packet feature processing is added to the delay required for detection caused by using session features. However, if this problem can be resolved, the intrusion detection performance of the NIDS can be dramatically improved by comprehensively using session and packet features.

The significant data needed to create a feature requires a correspondingly large storage space. Packet-feature-based NIDS collects sequentially received packets for each session up to a predetermined size and then inputs them to an ML model for classification. Therefore, past packets must be stored until sufficient packet data are gathered. Hence, creating an ML model where a packet-feature-based NIDS performs classification tasks incrementally every time packets are received and only stores intermediate classification results of small size and does not store packet data will significantly reduce the storage space required for NIDS.

The session-feature-based NIDS stores all packet data or packet headers for all packets in the session and analyzes them after the session is completed to determine their statistical characteristics, resulting in a high storage burden and long delay to confirm session termination, as explained above. Here, re-examining the characteristics of packet-feature-based NIDS indicates that some data from sessions can be used to achieve sufficiently high detection rates; thus, session features obtained from some initial session packets are likely to contain sufficient information. In addition, it is possible to fundamentally solve the delay problem of waiting for the maximum IAT until the end of a session. However, assuming that NIDS has improved to generate session features with only some initial session packets, the same storage space as the existing NIDS will be eventually used if all those packets still need to be stored; thus, storage space issues will remain unresolved again. Therefore, a new approach to creating session features that can address these constraints is urgently needed.

As session features primarily use statistical values that reflect the entire session, packets need not be stored if each session feature can be represented as a recurrent expression. With this approach, packet data can be discarded immediately after updating the existing session feature value every time a packet is received. Therefore, if both ideas can be implemented, the overhead for storing packet data can be fundamentally solved, even if the NIDS uses packet and session features simultaneously.

The proposed approach presents solutions to the two questions presented in the aforementioned motivation: 'Can classification be performed each time a packet is received?' and 'Can session features be generated in a recurrent manner?' Each solution is described in detail.

3.2. Incremental Classification Using Packet Data

In a typical deep learning model with fixed input shapes, it is not possible to perform a gradual classification of the sessions to which the received packets belong each time they are received [21,22]. However, deep learning models, such as recurrent neural network (RNN), gated recurrent unit (GRU), and long short-term memory (LSTM), address these constraints by generating cycles between nodes [23–26]. Thus, the deep learning model of the proposed NIDS performs classification with the LSTM classifier for every packet received, stores the results, and uses this result and newly received packets as LSTM inputs to progressively continue the classification for the session. Therefore, the required storage space can be significantly reduced because the packet data itself need not be stored and only the output result of the corresponding cell of the LSTM needs to be stored. Figure 2 shows a block diagram of the LSTM used in the proposed scheme for the incremental classification of sessions. When the proposed NIDS receives the first packet (i.e., Packet 1) of the session, it inputs packet 1 to the first cell (i.e., Cell 1) of the LSTM classifier. Afterwards, the hidden states and cell states of cell 1 are temporarily stored in the buffer until the next packet of the session (i.e., Packet 2) is received. When NIDS receives Packet 2, it inputs the values stored in the buffer and Packet 2 into Cell 2 to proceed with the next classification process, and the hidden states and cell states of Cell 2 are stored in the buffer again. In this way, classification of one session is gradually progressed, and the output value of the last cell (i.e., Packet *N*) is used as a feature to classify the session.

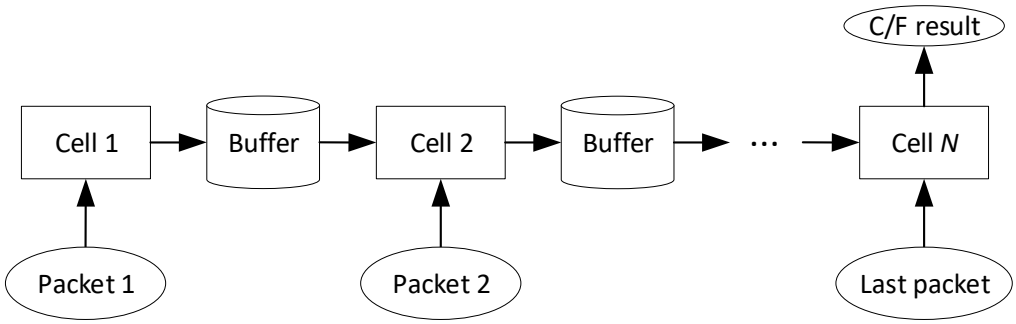


Figure 2. Proposed incremental classification for a session using LSTM.

3.3. Recurrent Session Feature Generation

For the feature set presented in ISCXIDS2012, the importance of each feature was measured using a random forest, and 15 features, including the top 10 features, were selected and analyzed to determine whether they could be expressed recursively. According to the analysis results shown in Table 4, although some states for feature generation need to be added, the features and state values for the current packet can be recurrently calculated through the features and state values for the previous packet. Using this approach, instead of collecting all packets and then creating session features, we update the state and features online as each packet is received and create the features shown in Table 1 when needed, eliminating the need to store any packets.

Table 4. Recurrently creating selected features without storing any packet data. Fifteen features are created recursively using seven states in addition to fifteen features. For example, for the Flow Duration feature, 'old Flow Duration + new Flow IAT' implies that a new Flow Duration value can be obtained by adding the updated Flow IAT value to the previous Flow Duration, when a new packet belonging to the current session is received.

Label	Type	Recurrent Expression
Last Flow Timestamp	State	current packet timestamp
Flow IAT	State	new Last Flow Timestamp—old Last Flow Timestamp
Fwd Last Timestamp	State	new current packet timestamp if the current packet is sent forwards
Bwd Last Timestamp	State	new current packet timestamp if the current packet is sent backwards
Fwd IAT	State	new Last Flow Timestamp—old Fwd Last Timestamp if the current packet is sent forwards
Flow IAT^2 Mean	State	(old Flow IAT^2 Mean $\times$ old Tot Pkts + new Flow IAT^2)/new Tot Pkts
Pkt Len^2 Mean	State	((old Tot Fwd Pkts + old Tot Bwd Pkts) $\times$ old Pkt Len^2 Mean + packet $size^2$)/(new Tot Fwd Pkts + new Tot Bwd Pkts)
Flow Duration	Feature	old Flow Duration + new Flow IAT
Tot Fwd Pkts	Feature	old Tot Fwd Pkts + 1 if the current packet is sent forwards
Tot Bwd Pkts	Feature	old Tot Bwd Pkts + 1 if the current packet is sent backwards
Flow IAT Mean	Feature	(old Flow IAT Mean $\times$ old Tot Pkts + new Flow IAT)/new Tot Pkts
TotLen Fwd Pkts	Feature	old TotLen Fwd Pkts + the packet size if the current packet is sent forwards
TotLen Bwd Pkts	Feature	old TotLen Bwd Pkts + the packet size if the current packet is sent backwards
Pkt Len Mean	Feature	(new TotLen Fwd Pkts + new TotLen Bwd Pkts)/(new Tot Fwd Pkts + new Tot Bwd Pkts)
Flow IAT Std	Feature	sqrt(new Flow IAT^2 Mean $-$ new Flow IAT $Mean^2$)
Fwd Pkts/s	Feature	new Tot Fwd Pkts/(new Flow Duration $-$ (new Last Flow Timestamp $-$ new Fwd Last Timestamp)
Flow Pkts/s	Feature	(new Tot Fwd Pkts + new Tot Bwd Pkts)/new Flow Duration
Fwd IAT Mean	Feature	(old Fwd IAT Mean $\times$ old Tot Fwd Pkts + new Fwd IAT)/new Tot Fwd Pkts if current packet is sent forwards
Pkt Len Std	Feature	sqrt(new Pkt Len^2 Mean $-$ new Pkt Len $Mean^2$)
Bwd Pkts/s	Feature	new Tot Bwd Pkts/(new Flow Duration $-$ (new Last Flow Timestamp $-$ new Fwd Last Timestamp)
Flow IAT Mean	Feature	(old Flow IAT Mean $\times$ old Tot Pkts + new Flow IAT)/new Tot Pkts
Flow IAT Max	Feature	max(old Flow IAT, new Flow IAT)

3.4. System Architecture

Essentially, the packet-feature-based NIDS uses only a few packets at the beginning of the session. Therefore, to use both the packet and session features, a session feature must be created using only the packets used by the packet-feature-based NIDS. If the number of packets used to create a packet feature and session feature is different, using more packets will eventually become a bottleneck in the feature creation speed.

Owing to varying packet sizes, even one packet cannot be classified by deep learning models, such as convolution neural network (CNN) and deep neural network (DNN), which can only process fixed input sizes. Therefore, a classifier consisting of a two-stage LSTM is required. The first-stage LSTM classifies packets of various sizes and inputs the result to a specific cell of the second-stage LSTM, enabling the second-stage LSTM classifier to classify sessions of various lengths.

Simultaneously, packets processed by LSTM update some features and state the values necessary for creating the rest of the session features. For the packet input to the last LSTM

cell, after updating the features and states, the session features for the session to which the current packet belongs are created. The generated session features are concatenated with the output of the two-stage LSTM and passed as input to the DNN to detect intrusion.

Figure 3 shows the overall architecture of the proposed NIDS. When packets belonging to a session are sequentially received, NIDS updates states and some feature values used to create session features. When the NIDS receives the *M*-th packet, the states and features are used to generate session features after updating, and the created session features are input to the DNN. At the same time, the received packet is divided into equal sizes as shown in Figure 3, and each partial packet is sequentially input to the first stage LSTM to create a packet feature to be used in the second stage LSTM. The generated packet feature is input to each cell of the second stage LSTM, and after the packet feature for the *M*-th packet is input, the output of cell *M* is input to another feature of the DNN. Now, the DNN determines whether the session is malicious by combining the two input features as inputs and using them for classification.

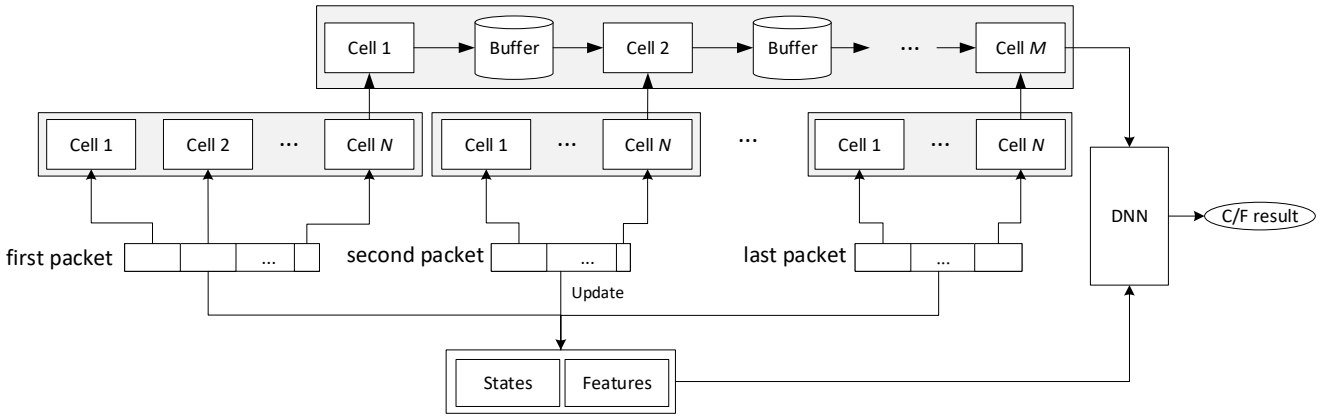


Figure 3. Proposed system architecture composed of two-stage LSTM and DNN.

4. Results

To accurately analyze the performance of the proposed scheme, we compared it with various existing deep-learning-based NIDS. In particular, the ISCXIDS2012 and CIC-IDS2017 datasets were used to verify the performance in various environments [9,10]; the characteristics of these datasets are listed in Tables 5 and 6, respectively. The fields dependent on a specific session, such as the source IP, destination IP, and source port, were removed from the packet data and session features to accurately train the ML models. In addition, representative deep learning algorithms, such as DNN, CNN, and HAST-I, were selected for comparison. Parameter settings for each algorithm are shown in Table 7. For performance analysis, we compared the performance of the proposed method and other algorithms by measuring the accuracy, precision, recall, F1-score, and confusion matrix. Each definition of metric is as follows:

- $\text{accuracy} = \dfrac{TP + TN}{TP + FP + FN + TN}$
- $\text{precision} = \dfrac{TP}{TP + FP}$
- $\text{recall} = \dfrac{TP}{TP + FN}$
- $\text{F1} - \text{score} = \dfrac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$

where, *TP*, *TN*, *FP*, and *FN* represent the true positive, true negative, false positive, and false negative, respectively. We also compared the proposed method and other NIDSs in terms of memory requirement and detection speed.

Table 5. Details of the ISCXIDS2012 dataset.

Number of features	81	
Number of classes	5	
Number of sessions	78,878	
Number of sessions for each class	Normal	22,382
	DDoS	21,702
	BruteForceSSH	18,145
	HTTPDoS	9487
	Infiltration	7162

Table 6. Details of the CIC-IDS2017 dataset.

Number of features	81	
Number of classes	11	
Number of sessions	123,236	
Number of sessions for each class	Benign	27,234
	DDoS	24,860
	DoS Hulk	20,419
	DoS GoldenEye	15,305
	PortScan	6459
	FTP-Patator	6267
	SSH-Patator	6029
	DoS slowloris	5256
	DoS Slowhttptest	4630
	Bot	3546
	Web Attack Brute Force	3231

Table 7. Parameter configuration for each NIDS.

Type	**Parameter**	**Value**
Proposed	Packet LSTM Unit	512, 256
	Session LSTM Unit	1024
	Session Unit	256, 128, 16
	Session Drop Out	0.1, 0.1, 0.1
	Activation	ReLU
CNN	Conv Unit	3, 3, 3, 3
	Kernel size	3, 3, 3, 3
	Unit	512, 256, 128
	Activation	ReLU
DNN	Unit	1024, 768, 512, 256, 128
	Density	0.5
	Activation	ReLU
HAST-I	Conv_unit	32, 64
	Kernel size	5, 5
	Pooling size	3, 3
	Unit	1024
	Activation	ReLU

4.1. Memory Requirement

Table 8 shows the memory size required to store the packet data used to create the feature. In the existing session-feature-based NIDS, the size of memory required to store packets for generating features is proportional to the average session length. On the other hand, the existing packet-feature-based NIDS uses some data for an initial fixed number of packets, so the required memory size is smaller than that of the session-feature-based

NIDS. The proposed NIDS minimizes the required memory area even though both feature types are used simultaneously. First, instead of using packet data to create session features, all features are created using only 17 states and some session features. Therefore, the proposed method always uses the same memory to create session features regardless of the session length. In addition, when generating packet features, only the current packet is used instead of several packets received so far, so only a fixed size of memory is used regardless of the session length. From Table 8, the existing session-based NIDS consumes 21,066 and 5791 bytes per session on average for the ISCXIDS2012 and the CIC-IDS2017, respectively. This means that the sessions in the ISCXIDS2012 are longer than the sessions in the CIC-IDS2017 and therefore use more memory. On the other hand, packet-feature-based NIDS requires 418 bytes per session in the CIC-IDS2017 instead of 357 bytes per session in the ISCXIDS2012. From this result, we find that ISCXIDS2012 contains more sessions, or six packets shorter than the CIC-IDS2017.

Table 8. Average memory size in bytes for storing the packet data needed to create the feature. Assume that only initial six packets are used for each session and only 100 bytes of each packet are used to create packet-based features.

Dataset	Session-Feature-Based	Packet-Feature-Based	Proposed
ISCXIDS2012	21,066	357	296 (196 + 100)
CIC-IDS2017	5791	418	296 (196 + 100)

The proposed NIDS creates two types of features simultaneously, using 196 bytes of memory per session for session features and only 100 bytes of memory per session for packet features. Therefore, since a total of 296 bytes of memory is used per session, the proposed NIDS consumes less memory than the existing session-based NIDS or packet-based NIDS, even though both features are used simultaneously. Above all, it is also a great advantage that the size of memory required per session is always fixed. It is advantageous for system design, such as being able to accurately calculate the memory required for the number of concurrent sessions that can be supported.

4.2. Detection Speed

Table 9 shows the number of packets required for intrusion detection per session between the existing session-feature-based NIDS and the proposed NIDS, where a smaller number of packets means faster detection speed. Although the number of packets required for detection varies depending on the dataset used, it shows that the proposed method uses a very small number of packets compared to the existing session-feature-based NIDS. In general, packet-feature-based NIDS is advantageous in increasing intrusion detection speed because it uses only some initial session packets. However, it should be noted that the proposed NIDS uses session features in addition to packet features. From Table 9, it is confirmed that the detection speed of the proposed NIDS can be greatly improved, unlike the existing session-feature-based NIDS, even though the proposed method uses the same session features to session-feature-based NIDSs. Due to the synergistic effect of packet and session features, the proposed NIDS can reduce the number of required packets effectively, increasing detection speed.

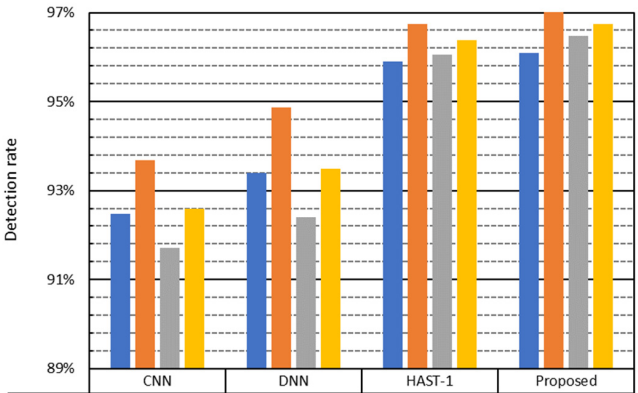
Table 9. The average number of packets required to detect an intrusion per session.

Dataset	Session-Feature-Based	Proposed
ISCXIDS2012	32.92	3.6
CIC-IDS2017	9.04	4.1

4.3. Intrusion Detection Accuracy

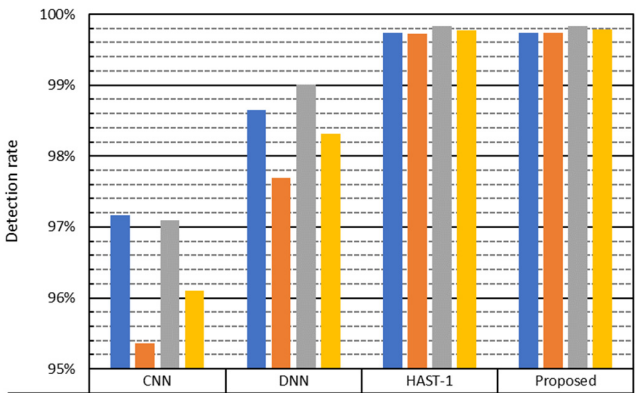
Figure 4 shows the intrusion detection rates of each algorithm for the datasets mentioned. As shown in the figure, the proposed scheme has the highest detection rate com-

pared with the existing NIDS. In particular, it is noteworthy that the proposed NIDS has the highest performance for all metrics, i.e., accuracy, precision, recall, and F1-score, among all the comparison algorithms. Considering that the proposed method detects intrusion without packet storage, in contrast to the existing methods that require a large memory size to store many packets to create features, the high detection accuracy of the proposed NIDS proves that it effectively mitigates the disadvantages of existing NIDSs while maintaining high detection accuracy.



	CNN	DNN	HAST-1	Proposed
Accuracy	92.48%	93.39%	95.91%	96.09%
Precision	93.68%	94.88%	96.74%	97.01%
Recall	91.70%	92.40%	96.06%	96.48%
F1-score	92.59%	93.49%	96.39%	96.74%

(**a**)



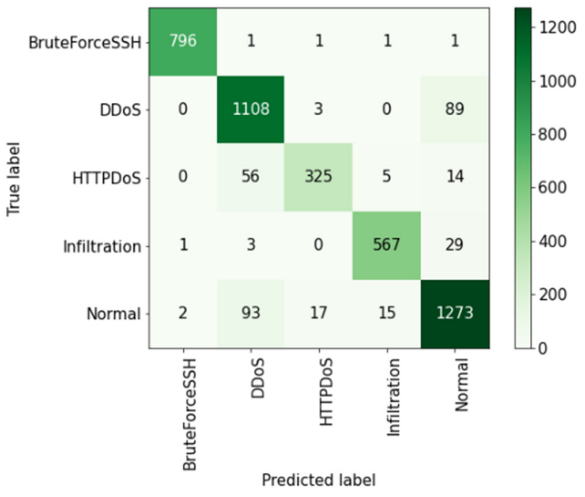
	CNN	DNN	HAST-1	Proposed
Accuracy	97.16%	98.65%	99.74%	99.74%
Precision	95.37%	97.69%	99.72%	99.73%
Recall	97.09%	99.01%	99.83%	99.84%
F1-score	96.10%	98.31%	99.77%	99.79%

(**b**)

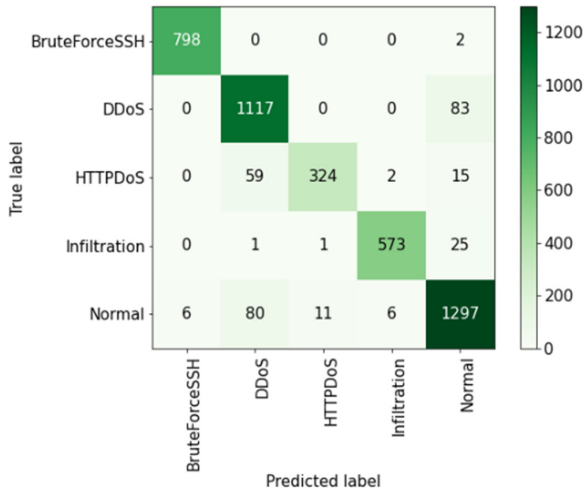
Figure 4. Comparison of intrusion detection rates for each NIDS using each dataset. (**a**) ISCXIDS2012; (**b**) CIC-IDS2017.

4.4. Confusion Matrix

Figures 5 and 6 show the confusion matrices for each algorithm for the two datasets. The confusion matrix is advantageous for analyzing detailed performance because it can analyze the performance of individual classes. When using ISCXIDS2012, the proposed method has a slightly lower detection rate for distributed denial-of-service (DDoS) than HAST-I but exhibits the highest performance for the other classes.



(**a**)



(**b**)

Figure 5. *Cont.*

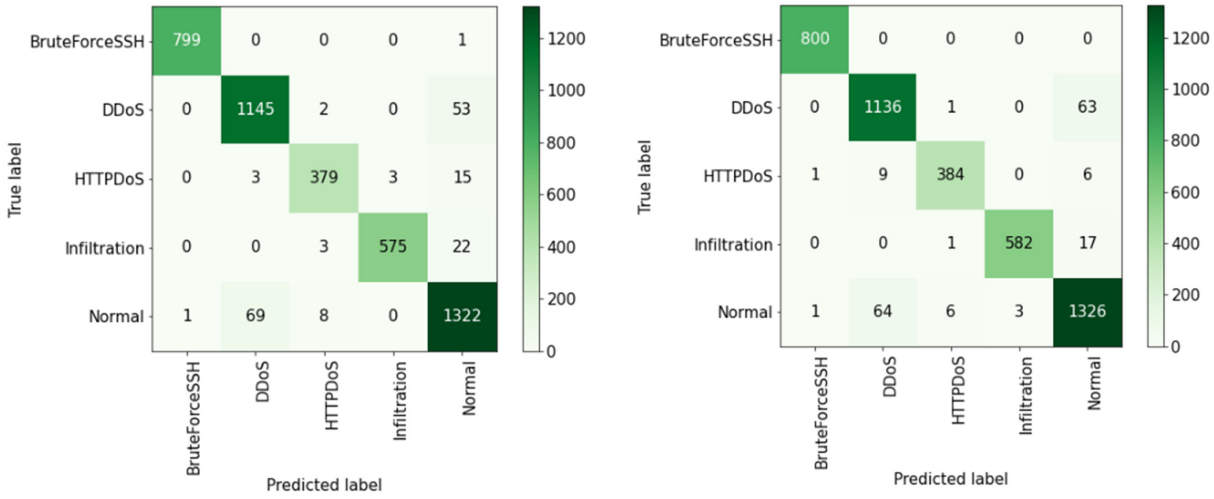


Figure 5. Confusion matrix for each NIDS using ISCXIDS2012. (**a**) CNN; (**b**) DNN; (**c**) HAST-I; (**d**) Proposed.

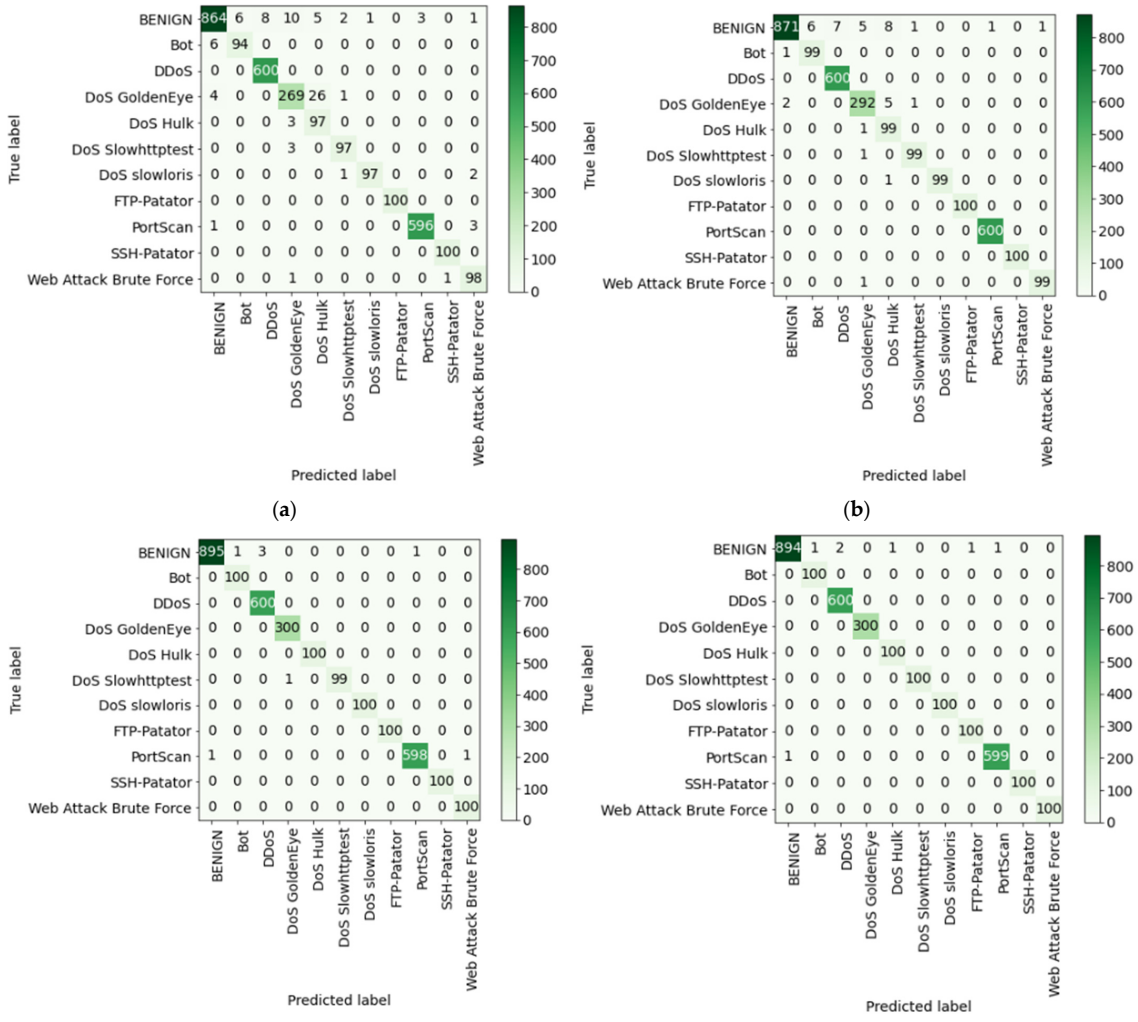


Figure 6. Confusion matrix for each NIDS using CIC-IDS2017. (**a**) CNN; (**b**) DNN; (**c**) HAST-I; (**d**) Proposed.

As shown in Figure 5, the DDoS class shows the lowest detection rate regardless of algorithm type. The reason is that DDoS attacks use many zombie PCs to attack one target with multiple sessions, so the characteristics are very different from other classes. Most NIDS today detect attacks based only on information about a single session, making them vulnerable to DDoS attacks that use multiple sessions. In this case, the additional use of characteristics for multiple sessions is helpful for detection.

In contrast, CIC-IDS2017 exhibits almost the same performance as HAST-I as shown in Figure 6. Therefore, these two confusion matrices confirm that the proposed NIDS has a high detection rate for the entire dataset but can also significantly improve the detection rate for individual classes.

CIC-IDS2017 has fewer noises than other datasets including the ISCXIDS2012. Therefore, compared to the ISCXIDS2012, the classification accuracy is higher regardless of the machine learning algorithm, so the margin for improving the accuracy is very small. Nevertheless, it shows that the proposed method is superior in that the proposed NIDS improves F1-scores by 3.67% points and 1.46% points, respectively, compared to DNN and CNN models. Since performance differences are evident for each classification model, it also shows that the CIC-IDS2017 is sufficient to be used for performance comparison.

5. Conclusions

The proposed NIDS does not need to store the received packets for feature creation; therefore, in contrast to the existing ML-based As, the amount of memory consumed for processing each active session is minimal. Because the same memory can support a greater number of concurrent sessions than other NIDS, the insufficient NIDS processing capacity owing to the recent increase in network traffic can be significantly improved. Above all, it is a significant advantage that the intrusion detection performance can be improved compared to the existing NIDS, despite the small memory footprint. As network attacks diversify and zero-day attacks become frequent in an environment where the amount of network traffic increases drastically, NIDS faces technically significant challenges in simultaneously improving processing capacity, speed, and detection accuracy. Hence, the proposed NIDS is expected to significantly aid in solving these problems.

As the proposed NIDS operates optimally when packets within a session are received in order, the amount of memory used for sorting increases when there are many out-of-order packets. The proposed NIDS is designed under the assumption that session packets are received without loss. However, some packets may be lost in real networks. It is expected that packet loss causes a negative impact on intrusion detection performance. The weaknesses of the proposed NIDS will be addressed through future research, and we expect that the proposed NIDS will be fully applied to an actual network.

Author Contributions: J.H. and W.P. wrote the paper and conducted the research. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) NRF2022R1A2C1011774.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets utilized in this paper are ISCXIDS2012 dataset (https://www.unb.ca/cic/datasets/ids.html (accessed on 1 February 2023) and CIC-IDS2017 dataset (https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 1 February 2023).

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

References

1. Kruegel, C.; Toth, T. Using decision trees to improve signature-based intrusion detection. In Proceedings of the 2003 International Workshop on Recent Advances in Intrusion Detection, Pittsburgh, PA, USA, 8–10 September 2003; pp. 173–191. [CrossRef]
2. Wu, S.X.; Banzhaf, W. The use of computational intelligence in intrusion detection systems: A review. *Appl. Soft Comput.* **2010**, *10*, 1–35. [CrossRef]
3. Ektefa, M.; Memar, S.; Sidi, F.; Affendey, L.S. Intrusion detection using data mining techniques. In Proceedings of the 2010 Information Retrieval & Knowledge Management (CAMP), Selangor, Malaysia, 16–18 May 2010; pp. 200–203. [CrossRef]
4. Bilge, L.; Dumitras, T. Before we knew it: An empirical study of zero-day attacks in the real world. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 833–844. [CrossRef]
5. Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* **2017**, *6*, 1792–1806. [CrossRef]
6. Cisco. Cisco Annual Internet Report (2018–2023). White Paper, 2017–2022. Available online: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed on 15 January 2023).
7. Li, L.; Yu, Y.; Bai, S.; Hou, Y.; Chen, X. An Effective Two-Step Intrusion Detection Approach Based on Binary Classification and k-NN. *IEEE Access* **2017**, *6*, 12060–12073. [CrossRef]
8. Tavallaee, M.; Bagheri, E.; Lu, W.; Ghorbani, A. A Detailed Analysis of the KDD CUP 99 Data Set. In Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), Ottawa, ON, Canada, 8–10 July 2009. [CrossRef]
9. Shiravi, A.; Shiravi, H.; Tavallaee, M.; Ghorbani, A.A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.* **2012**, *31*, 357–374. [CrossRef]
10. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In Proceedings of the 2018 4th International Conference on Information Systems Security and Privacy (ICISSP), Madeira, Portugal, 22–24 January 2018. [CrossRef]
11. Soheily-Khah, S.; Marteau, P.; Béchet, N. Intrusion Detection in Network Systems Through Hybrid Supervised and Unsupervised Machine Learning Process: A Case Study on the ISCX Dataset. In Proceedings of the 1st International Conference on Data Intelligence and Security (ICDIS), South Padre Island, TX, USA, 8–10 April 2018; pp. 219–226. [CrossRef]
12. Chen, C.; Xu, X.; Wang, G.; Yang, L. Network intrusion detection model based on neural network feature extraction and PSO-SVM. In Proceedings of the 7th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 15–17 April 2022. [CrossRef]
13. Jiawei, D.; Kai, Y.; Zhentao, H.; Lingjie, H.; Lei, H.; Haixia, Y. Research on Intrusion Detection Algorithm Based on Optimized CNN-LSTM. In Proceedings of the International Conference on Networking and Network Applications (NaNA), Urumqi, China, 18–21 August 2022. [CrossRef]
14. Eddy, W. Transmission Control Protocol (TCP). Available online: https://www.rfc-editor.org/info/rfc9293 (accessed on 15 January 2023).
15. Samsung Secui. BlueMax NXG. Available online: https://www.secui.com/en/network/bluemaxngf (accessed on 15 January 2023).
16. Sharafaldin, I.; Lashkari, A.H.; Hakak, S.; Ghorbani, A.A. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In Proceedings of the IEEE 53rd International Carnahan Conference on Security Technology, Chennai, India, 1–3 October 2019. [CrossRef]
17. Lashkari, A.H.; Draper-Gil, G.; Mamun, M.; Ghorbani, A.A. Characterization of Tor Traffic Using Time Based Features. In Proceedings of the 2017 3rd International Conference on Information System Security and Privacy, SCITEPRESS, Porto, Portugal, 19–21 February 2017. [CrossRef]
18. Drapper-Gil, G.; Lashkari, A.H.; Mamun, M.; Ghorbani, A.A. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In Proceedings of the 2016 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016), Rome, Italy, 19–21 February 2016; pp. 407–414. [CrossRef]
19. Sahu, S.; Mehtre, B.M. Network intrusion detection system using J48 Decision Tree. In Proceedings of the 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Kochi, India, 10–13 August 2015; pp. 2023–2026. [CrossRef]
20. Description of Kyoto University Benchmark Data. Available online: https://www.takakura.com/Kyoto_data/BenchmarkData-Description-v5.pdf (accessed on 13 January 2023).
21. Gu, J.; Zhu, M.; Zhou, Z.; Zhang, F.; Lin, Z.; Zhang, Q.; Breternitz, M. Implementation and evaluation of deep neural networks (DNN) on mainstream heterogeneous systems. In Proceedings of the 2014 5th Asia-Pacific Workshop on Systems (APSys '14), Beijing, China, 25–26 June 2014; pp. 1–7. [CrossRef]
22. Valueva, M.V.; Nagornov, N.N.; Lyakhov, P.A.; Valuev, G.V.; Chervyakov, N.I. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Math. Comput. Simul.* **2020**, *177*, 232–243. [CrossRef]
23. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. *Learning Internal Representations by Error Propagation*; Tech. rep. ICS 8504; Institute for Cognitive Science, University of California: San Diego, CA, USA, 1985.
24. Cho, K.; van Merrienboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In Proceedings of the 2014 8th Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 24 October 2014. [CrossRef]

25. Fawcett, T. An Introduction to ROC Analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. [CrossRef]
26. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]