electronics

Article

Integrated Feature-Based Network Intrusion Detection System Using Incremental Feature Generation

Taehoon Kim 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: Machine learning (ML)-based network intrusion detection systems (NIDSs) depend entirely on the performance of machine learning models. Therefore, many studies have been conducted to improve the performance of ML models. Nevertheless, relatively few studies have focused on the feature set, which significantly affects the performance of ML models. In addition, features are generated by analyzing data collected after the session ends, which requires a significant amount of memory and a long processing time. To solve this problem, this study presents a new session feature set to improve the existing NIDSs. Current session-feature-based NIDSs are largely classified into NIDSs using a single-host feature set and NIDSs using a multi-host feature set. This research merges two different session feature sets into an integrated feature set, which is used to train an ML model for the NIDS. In addition, an incremental feature generation approach is proposed to eliminate the delay between the session end time and the integrated feature creation time. The improved performance of the NIDS using integrated features was confirmed through experiments. Compared to a NIDS based on ML models using existing single-host feature sets and multi-host feature sets, the NIDS with the proposed integrated feature set improves the detection rate by 4.15% and 5.9% on average, respectively.

Keywords: integrated feature set; machine learning; network intrusion detection system; incremental feature generation

Citation: Kim, T.; Pak, W. Integrated Feature-Based Network Intrusion Detection System Using Incremental Feature Generation. *Electronics* **2023**, *12*, 1657. https://doi.org/10.3390/electronics12071657

Academic Editor: Simeone Marino

Received: 7 March 2023
Revised: 24 March 2023
Accepted: 29 March 2023
Published: 31 March 2023

1. Introduction

At present, network intrusions are highly diverse and sophisticated. Therefore, it is becoming increasingly difficult to detect them accurately. To improve the accuracy of network intrusion detection, several studies have been conducted, using various technologies. In particular, as machine learning (ML) has evolved significantly, many network intrusion detection systems (NIDSs) that use ML have recently been proposed [1–3]. Unlike early ML-based NIDSs, which mainly used simple ML models, complex deep learning models have been recently used for network intrusion detection. Unlike conventional pattern-based NIDSs, deep learning-based NIDSs demonstrate a robust detection performance against detection evasion attacks that partially modify network intrusion methods and achieve a high detection performance against zero-day attacks, which are new and previously unknown [4–6]. Therefore, deep learning technology is the most important core technology for overcoming network intrusions.

However, several important problems must be solved to implement a deep-learning-based NIDS. First, a large dataset consisting of many intrusions and normal traffic is needed to train a deep learning model. Recently, various research institutes have been steadily releasing datasets, including the latest intrusion traffic. Studies to remove the bias of datasets generated from specific sites are also being conducted extensively. Therefore, the problem of the quantity of the datasets required to train the deep learning model was significantly alleviated. The second problem is the design of a deep learning model.

The most critical factor in the design is determining which data characteristics are used as learning features. The detection performance of the NIDS and the complexity of the deep learning model differ depending on the feature. However, unlike learning datasets, relatively few studies have been conducted to solve this problem. In an NIDS, it is common to use a feature that reflects a session's overall characteristics rather than a feature for each packet of data [7]. These session features were commonly used in the early days, starting with those presented in the KDD99 dataset [8]. However, the session feature presented in the KDD99 dataset is costly because sessions belonging to multiple hosts (sessions created in multiple hosts or created in one host and having multiple destinations) must be analyzed simultaneously to create the feature.

The session features used in the ISCX2012 dataset, later released by the University of New Brunswick (UNB), are partially similar to the session features presented in the KDD99 dataset [9]. However, its fundamental difference from the KDD99 dataset is that the ISCX2012 dataset only comprises features that can be created by analyzing sessions belonging to a single host (sessions created between the same single source and the same single destination). Therefore, generating session features in the ISCX2012 dataset by analyzing the network traffic is much easier than generating session features in the KDD99 dataset. However, little research has been conducted on the effects of certain session features on intrusion detection performance. Therefore, in order to improve the accuracy of existing ML-based NIDSs, we need to answer the following questions: first, which features are most suitable for ML models used in NIDSs? Second, can those features be generated without significant overhead so that they can be applied to existing NIDSs?

To find the answers in this study, experiments were conducted and analyzed to determine a method of using session features that can increase detection performance, focusing on session features. In addition, we propose a feature set that can further improve the detection performance of existing session-feature-based NIDSs. Finally, we introduce an incremental generation method to build new feature sets from the network traffic in semi-real time. Our contributions are as follows.

- We propose a unique integrated feature that combines single-host and multi-host features to significantly improve the detection accuracy of existing NIDSs. Through extensive experiments on features, we present the most suitable feature for ML-based NIDSs.
- We present an incremental generation algorithm to build integrated features in real-time without significant overhead. Although the integration feature can improve the classification accuracy of an NIDS the most, it is impossible to apply it to NIDSs due to the high overhead when it is generated in the existing algorithms. To solve this problem, we present a very lightweight, real-time feature generation algorithm which is totally different from the existing algorithms.

The remainder of this study is structured as follows. Section 2 explains the features used in previous studies, and Section 3 presents the new feature sets and progressive generation methods. Section 4 analyzes the performance of the NIDS by applying various ML models to the existing session feature set, including the proposed feature set. Finally, Section 5 presents the conclusions of this study.

2. Existing Work

The feature sets that are widely used in recent ML-based NIDSs can be divided into session features that are determined by analyzing traffic belonging to the entire session rather than individual packets and packet features that are created from the packet data. The session features can be further classified into single-host features, which are created by analyzing sessions between a single source and single destination, and multiple-host features, which can create sessions between a single source and multiple destinations or multiple sources and single destinations.

The well-known datasets that use single-host features are the ISCX2012, CIC-IDS2017, and CSE-CIC-IDS2018 datasets, published by the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick (UNB) [10,11]. Single-host features are easier to create because only the traffic from that session is required to create features for each session. In general, to create a session feature, the traffic of each session must be collected and analyzed from the beginning to the end of the session. Thus, the memory complexity for generating a session feature is $\theta(n)$, where n is the number of packets in the session. Recently, a new approach for generating a single in-line host feature has been proposed. It has been proven that a session feature can be gradually created by updating some data fields whenever each packet is received [12]. In this method, the memory complexity is $\theta(f)$ (where f is the number of features), which can significantly reduce complexity compared to the existing method and can generate session features immediately after the session is terminated in semi-real time by minimizing the feature extraction time. This indicates that it is possible to extend the non-real-time NIDS to a real-time network intrusion prevention system (NIPS).

Contrary to the single-host feature, the multi-host feature is created using all sessions with the same destination or the same source among sessions created within a specific period. Therefore, because many sessions involved in creating a session feature must be considered simultaneously, both the memory and time complexity are more significant than the complexity of a single-host feature. Due to this complexity, multi-host features are not commonly used at present. However, the multi-host feature contains valuable information for detecting distributed attacks using multiple zombie hosts. Because distributed attacks are increasingly common, the importance of conducting research on methods of creating multi-host features in real time at a low cost is growing. In particular, if a method for generating multi-host features in real time is found, it is expected that it will be possible to detect distributed attacks accurately in real time. The following table characterizes each feature. Single-host features are often used because they are simpler to create than multi-host features. However, single-host and multi-host features have different information for the traffic; therefore, if they are used simultaneously, they can compensate for each other's shortcomings. Therefore, in order to use both features simultaneously, it is important to develop a means to efficiently generate multi-host features in real time while minimizing resource usage. Each feature set type and corresponding dataset are described in detail in Table 1.

Table 1. Comparison for single-host and multi-host features.

Feature Type	Pros.	Cons.
Single-host	It can be created in real time. It consumes fewer resources for generation.	It includes insufficient information about attacks using multiple hosts.
Multi-host	It contains useful information to detect multi-host-based attacks.	It requires huge resources to generating features.

2.1. Single-Host Feature

The key to ML-based NIDS research is creating related datasets. However, this task requires considerable effort and time. When an ML-based NIDS was proposed in the early days, only limited datasets could be used practically. The CIC has recently provided various datasets necessary for network security research, as shown in Table 2. Therefore, most ML-based NIDS studies use CIC datasets.

The CIC creates session features using a self-developed tool called CICFlowMeter [13,14]. Table 3 shows a part of the feature set in which CICFlowMeter v3 is generated. Note that the features created by CICFlowMeter were created by analyzing packets in all sessions generated between a single source and a single destination. For example, the main features created are the number of packets transmitted and received within one TCP session, or the average size of packets within one session. The total number of feature sets was 80. Overall, these features

can be further divided into two types. An intra-flow feature is obtained by analyzing only one session, and an inter-flow feature is obtained by analyzing several sessions simultaneously.

Table 2. Datasets provided by CIC [12].

No	Name	Description
1	CIC-DDoS2019	DDoS Evaluation Dataset
2	CSE-CIC-IDS2018	IPS/IDS dataset on AWS
3	CIC-IDS2017	Intrusion Detection Evaluation Dataset
4	ISCX IDS2012	Intrusion Detection Evaluation Dataset

Table 3. Partial feature set generated by CICFlowMeter. IAT—inter-arrival time.

No	Name	Description	Type
1	Duration	Flow duration.	Intra-flow
2	Total forward packets	Total packets in the forward direction.	
3	Total backward packets	Total packets in the backward direction.	
4	Total forward size	Total size of packet in forward direction.	
5	Max forward size	Maximum size of packet in forward direction.	
6	Min forward size	Minimum size of packet in forward direction.	
7	Average forward size	Average size of packet in forward direction.	
8	Forward IAT standard deviation	Standard deviation size of packet in forward direction.	
9	Max backward size	Maximum size of packet in backward direction.	
10	Min backward size	Minimum size of packet in backward direction.	
11	Average backward size	Average size of packet in backward direction.	
12	Backward IAT standard deviation	Standard deviation size of packet in backward direction.	
...	...	...	
77	Average inter-flow time	Average time between two flows.	Inter-flow
78	Inter-flow time standard deviation	Standard deviation time between two flows.	
79	Max inter-flow time	Maximum time between two flows.	
80	Min inter-flow time	Minimum time between two flows.	

A dump file of packets is saved to create a single, host-based dataset from CIC, and the packet data for each session are then analyzed with CICFlowmeter to create a feature. This process is impossible to perform in real time within the NIDS because the memory usage is high. Therefore, the NIDS generates features in non-real time for terminated sessions and performs intrusion detection using an ML classifier. In a recent study, a method was proposed to create and update meta-feature values to create session features whenever packets are received from an NIDS without CICFlowmeter and to create single-host features without the high time and space complexity through them immediately after session termination. By using this method, it is possible to create features for sessions in near-real time in NIDSs such that the delay from session termination to detection can be minimized, solving the biggest drawback of the NIDS using existing single-host-based datasets.

This approach is called incremental feature generation (IFG) [15]. In Figure 1, the basic structure of the IFG is presented. The received packets are stored and processed independently according to their direction. This information is updated each time a packet is received, and the entire single-host feature is updated based on this information.

Therefore, unlike the method for analyzing session traffic after a session is terminated, a feature is created immediately after a session is terminated.

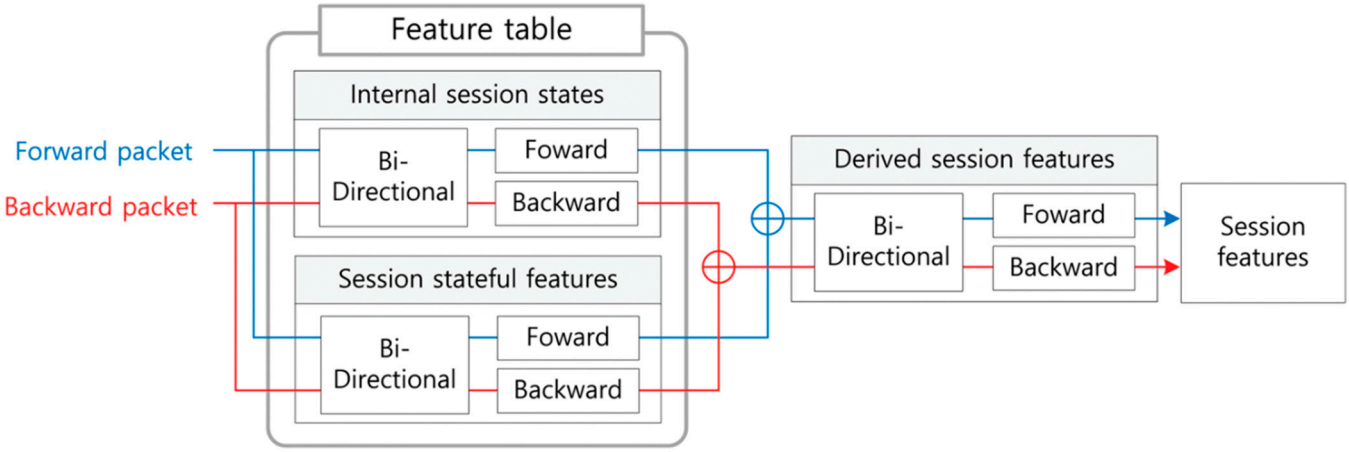


Figure 1. IFG block diagram for single-host feature creation.

2.2. Multi-Host Feature

The KDD99 and Kyoto2016 datasets contain single-host features similar to those created by CIC [16,17]. However, these datasets also include multi-host features with distinct characteristics from the CIC datasets. Higher computational complexity and memory complexity are required to create a multi-host feature than existing single-host features. As the KDD99 and Kyoto datasets are very similar, only the Kyoto dataset is described here.

The feature sets used in the Kyoto2016 dataset are as follows [17]. Excluding session-dependent fields, it consists of 11 single-host features and 7 multi-host features as shown in Table 4. Although it consists of a very small number of features compared to the 80 features of the dataset created by CIC, it shows a high intrusion detection success rate with multi-host features.

Table 4. Partial feature set used in the Kyoto dataset.

No	Feature	Description	Type
1	Source bytes	The total number of data bytes transmitted by the source IP address.	Multi host
2	Destination bytes	The total number of data bytes transmitted by the destination IP address.	
3	Dst host count	Among the past 100 connections whose destination IP address is the same as that of the current connection, the total number of connections whose source IP address is the same as that of the current connection.	
4	Dst host srv count	Among the past 100 connections whose destination IP address is the same as that of the current connection, the total number of connections whose service type is also the same as that of the current connection.	
5	Dst host same src port rate	The percentage of connections whose source port is the same as that of the current connection in Dst host count feature.	
6	Dst host serror count feature rate	The percentage of connections that have “SYN” errors in Dst count feature.	
7	Dst host srv serror rate	The percentage of connections that “SYN” errors in Dst host srv count feature.	

Table 4. *Cont.*

No	Feature	Description	Type
8	Duration	The length of the connection in seconds.	Single host
9	Service	The service type of the connection.	
10	Count	The total number of connections whose source IP address and destination IP address are the same as those of the current connection in the past two seconds.	
11	Same srv rate	The percentage of connections to the same service among the sessions in Count feature.	
12	Serror rate	The percentage of connections that have "SYN" errors among the sessions in Count feature.	
13	Srv serror rate	The percentage of connections that have "SYN" errors among the sessions in Srv count feature.	
14	Flag	The state of the connection.	
15	IDS detection	Indicates whether IDS triggered an alert for the connection.	
16	Malware detection	Indicates whether malware was observed in the connection.	
17	Ashula detection	Indicates whether shellcodes and exploit codes were used in the connection.	
18	Start Number	Indicates when the session was began.	
19	Source IP Address	Indicates the source IP address of the session.	Session specific
20	Source Port Number	Indicates the source port number of the session.	
21	Destination IP Address	Indicates the source IP address of the session.	
22	Destination Port Number	Destination Port of the session.	

2.3. Packet Feature

As mentioned above, because the session features are created by analyzing the packets received from the first to the last packet of each session, the features are inevitably created after the intrusion ends. In contrast, an NIDS using packet features collects a certain number of packet data and uses them directly as features. To use the session feature, it is necessary to decide the traffic characteristics that should be used as a feature in advance; however, the ML model using the packet feature does not require such complicated preliminary work. Instead, deep learning technologies, such as CNN, are essential because meaningful data must be obtained directly from the packet data [4]. The most severe problem with the packet feature is that because each byte of packet data is used as one feature, many features are generated, making real-time processing impossible. As one-hot encoding is applied to each byte, 100-byte packet data are converted into 25,600 features. Moreover, significantly high processing power and time are required to apply the deep learning algorithm to a dataset of many features. Therefore, the process of generating packet features using HAST-IDS, a representative NIDS that uses packet features, as shown in Figure 2, is applied [4]. After sequentially collecting packets of a specific size from the first packet, each byte value was expanded by one-hot encoding to create a packet feature.

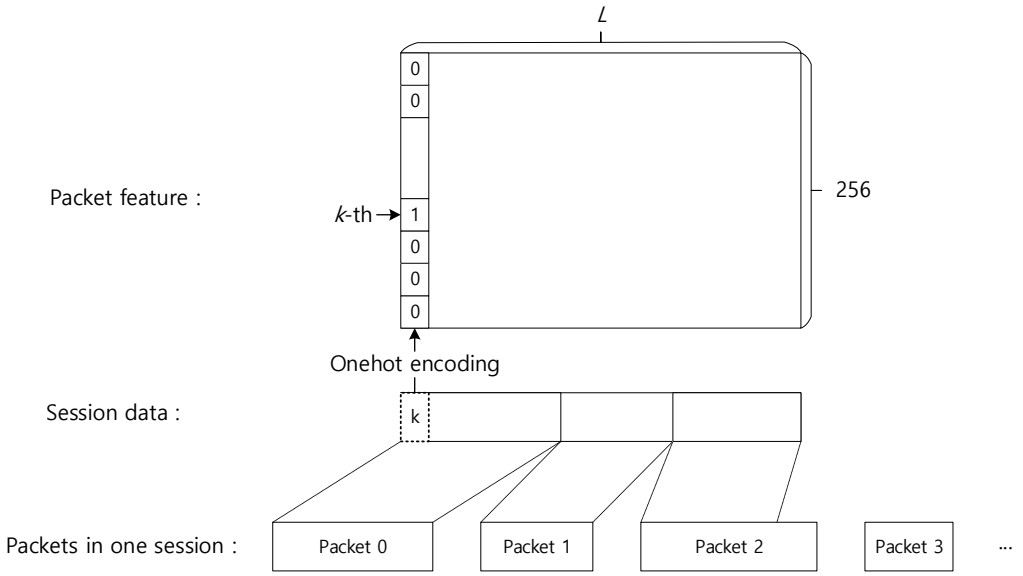


Figure 2. Process of generating packet features used in HAST-IDS.

3. Integrated Session Feature-Based NIDS

3.1. Incremental Session Feature

A new feature is proposed to overcome the disadvantages of the existing single-host or multi-host features. The proposed method removes duplicate features by integrating existing single-host and multi-host features. The integration feature also suggests an incremental generation method that can be created without delay when a session is terminated. Thus, this study did not consider packet features because it is impossible to generate them in real time. The number of fields in the unified feature is such that the features that overlap with the existing two features are duration and service; the source IP, destination IP, and source port vary according to sessions, and are thus excluded from the integrated features. Therefore, the total number of features was 97, as is shown in Table 5.

Table 5. Feature set size.

Feature Set	Integrated	ISCX2012	Kyoto2016
Feature size	97	80	22

To create a feature similar to the existing method, the packet is saved whenever a packet is received. When the session is terminated, the feature is created from all packet data using Zeek and CICFlowmeter [17]. However, in this case, a large amount of calculation and memory is consumed after the session is terminated. To improve this, the method of incrementally creating the existing single-host session feature was expanded. Using this method, both single-host and multi-host session features can be created incrementally. The incremental feature-creation method is illustrated in Figure 3. The structure to create the existing single-host session feature is extended to include a structure that stores the last 2 s of all sessions with the same SIP and counts the number of sessions with the same service and SYN error. This gradually creates all necessary single-host session features.

In addition, a new structure for creating a multi-host session feature was added. All sessions with the same DIP are implemented with a window that stores only the latest 100 sessions and a two-second window that stores sessions that occurred during the last two seconds. The number of sessions with the same SIP and SYN errors was calculated in real-time. This way, the multi-host session feature can be created without delay when the session is terminated.

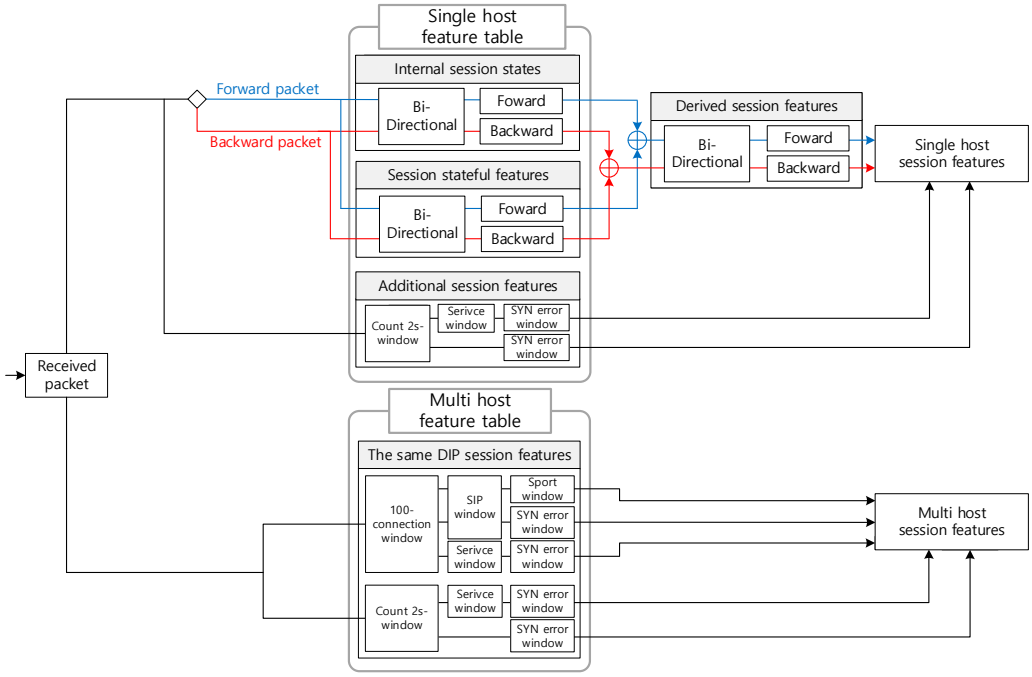


Figure 3. Block diagram for incremental generating integrated features from data traffic.

3.2. Incremental Feature Generation

Let us describe how to incrementally generate session features in detail. Due to lack of space, we will show how to obtain features only for the Kyoto2016 dataset, which is more difficult than obtaining features for ISCX2012. As shown in Figure 3, to generate single-host features and multi-host features, the proposed NIDS must manage sessions using a session window of the past 2 s regardless of DIP and the last 100 sessions for each DIP. To manage many sessions in real time, the proposed method uses a linear queue to manage active sessions for the past 2 s, as shown in Figure 4. Therefore, if the end time is over 2 s, the session entry is deleted from the linear queue. In addition, to manage the last 100 sessions for each DIP, the proposed NIDS uses a hash table with the DIP as the key and circular queues referenced by the hash entries.

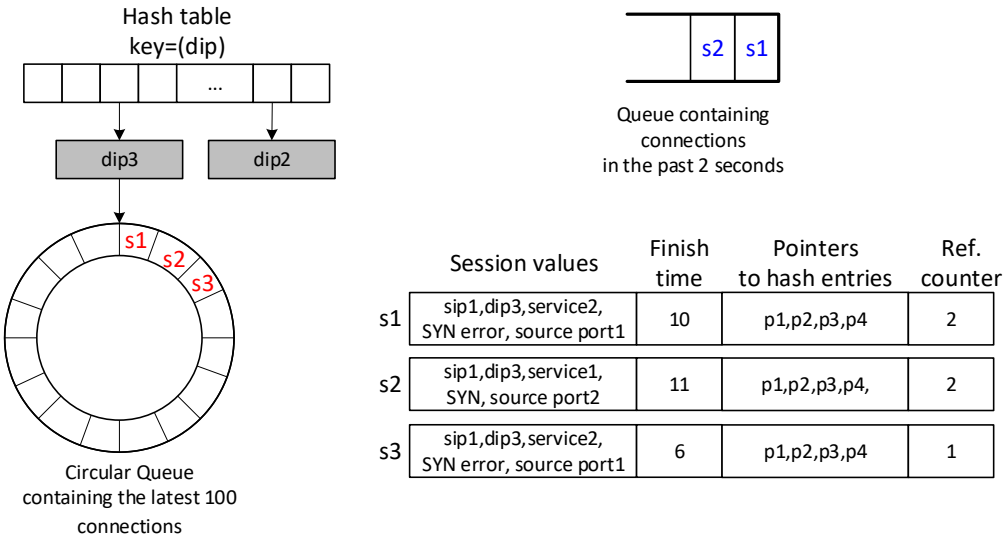


Figure 4. Examples of two queues and corresponding session entry structures. The circular queue contains three sessions while the linear queue contains two sessions.

A session entry consists of the session information (i.e., SIP, DIP, service, SYN error status, and source port number) in addition to a session end time, a pointer list to hash table entries, and a reference counter. Since the proposed scheme uses four hash tables, the pointer to the k-th hash table entry is denoted by pk (k = 1, 2, . . . , 4). The reference counter indicates how many queues the session is stored in. Since the proposed method uses two queues to manage sessions, the reference counter can be set to 0, 1, and 2.

When a session is created, a session entry is created and stored in the circular and linear queues. For the sessions stored in these two queues, four additional hash tables are used to maintain the counting values needed to generate the session features. Figure 5 shows the structure of the hash tables used by the proposed method. The pk of the session entries are used to quickly update the four hash tables when the existing session entry is deleted.

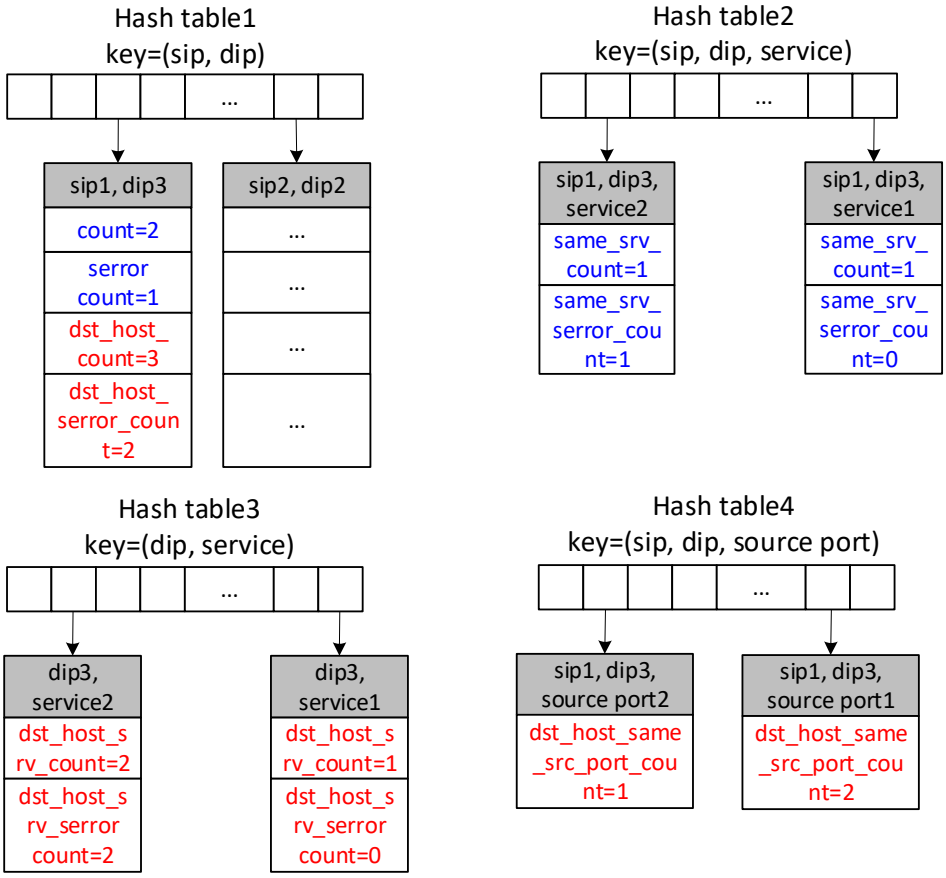


Figure 5. Four hashes to maintain statistics needed to generate multi-host session features in real-time. The blue and red colors represent values associated with sessions stored in the linear queue and the circular queue, respectively.

As the proposed NIDS uses each hash table similarly, let us describe only how hash table1 works. Each entry in hash table1 contains a "count" field, which stores the number of sessions with the same SIP and DIP for the sessions stored in the linear queue, and a "serror_count" field, which stores the number of sessions with SYN errors among them. It also contains a "dst_host_count" field, which stores the number of sessions with the same SIP and DIP for the sessions stored in the circular queue, and a "dst_host_serror_count" field, which stores the number of sessions with SYN errors. These counters are updated whenever a new session is created.

We will only describe how to obtain the multi-host session feature, which is more complicated than the single-host session feature. Whenever a new session is created or terminated, the circular queue, linear queue, or four hash tables can be updated. Then, if the proposed method needs the multi-host session feature for a particular session, the

features are computed as shown in Table 6. It finds a session entry for the currently received session at first. Then, using p*k*, it accesses the entries in each hash table directly. Eventually, all the features in Table 6 can be obtained with a time complexity of θ(4). For example, for session S, the feature 'Same srv rate' is given by

$$\text{Same srv rate} = \frac{\text{same\_srv\_count}}{\text{count}} \quad (1)$$

where 'same_srv_count' for session S can be read via the p2 pointer from the session entry for session S. The count can also be read through the p1 pointer, making it very fast to calculate the feature.

Table 6. List of multi-host features and their calculation expressions. hash table *k* (f) means the value of the field f in the entry of the *k*-th hash table matching with the given session.

Feature	Feature Calculation
Count	hash table1(count)
Same srv rate	hash table2(same_srv_count)/hash table1(count)
Serror rate	hash table1(serror_count)/hash table1(count)
Srv serror rate	hash table2(same_srv_serror_count)/hash table2(same_srv_count)
Dst host count	hash table1(dst_host_count)
Dst host srv count	hash table3(dst_host_srv_count)
Dst host same src port rate	hash table4(dst_host_same_src_port_count)/hash table1(dst_host_count)
Dst host serror rate	hash table1(dst_host_serror_count)/hash table1(dst_host_count)
Dst host srv serror rate	hash table3(dst_host_srv_serror_count)/hash table3(dst_host_srv_count)

4. Performance Evaluation

4.1. Experiment Environment

We evaluated the performance of the feature-creation method described above and determined the effectiveness of the proposed feature. The feature set compares the performance of the proposed integrated feature using the single-host feature created using CIC FlowMeter with the multi-host feature used in the Kyoto 2016 dataset, which was created using Zeek. The dataset used for the performance analysis was CIC-IDS2017, published by CIC. The size of the dataset used for performance analysis is listed in Table 7. The entire dataset was randomly divided into 60% and 40% to create the training and test datasets, respectively.

The algorithms used for performance comparison are the decision tree (DT), naïve Bayes (DTNB), DT with *k*-NN (DTKNN), synthetic minority over-sampling technique (SMOTE) + random forest (RF), support vector machine (SVM), and 1D convolutional neural networks (1D-CNN) algorithms, which are widely used in NIDSs [4,18–24]. The performance metrics analyzed were accuracy, precision, recall, and F1-score. The definitions are as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \quad (2)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (3)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4)$$

$$\text{F1-score} = \frac{TP}{TP + 0.5(FP + FN)} \tag{5}$$

Here, *TP*, *TN*, *FP*, and *FN* represent the true positive, true negative, false positive, and false negative, respectively.

Table 7. Dataset size of CIC-IDS2017.

Class	Total Size	Training Size	Test Size
Benign	978,480	587,088	391,392
FTP-Patator	2411	1447	964
SSH-Patator	1829	1097	732
DoS Hulk	43,442	26,065	17,377
DoS Slowhttp	39,738	23,843	15,895
DoS Slowloris	31,268	18,761	12,507
Web Bruteforce	881	529	352
Bot	1358	815	543
DDoS	57,623	34,574	23,049
Portscan	95,382	57,229	38,153
Total	1,252,412	751,448	500,964

4.2. Detection Rate

The detection performance of each machine learning algorithm for each feature is presented in Figure 6. Based on the F1-scores, the performance of each algorithm was measured as high in the order of 1D-CNN, DTNB, SMOTE + RF, DT, DTKNN, and SVM. Regardless of the type of algorithm, including traditional ML or recent deep-learning models, the performance of the NIDS using single-host features is 1.75% higher on average compared with that of the NIDS using multi-host features, based on the F1-scores. In addition, the NIDS using the integrated feature created by combining the two session features has a higher performance than the NIDS using the multi- or single-host features, specifically, 5.9% higher than the multi-host feature and 4.15% higher than the single-host feature. Considering that the difference between the single-host and multi-host features is 1.75%, the performance improvement is substantial. Excluding SMOTE + RF, the performance increases in the order of multi-host features, single-host features, and integrated features for all metrics: accuracy, precision, recall, and F1-score. In the case of DTKNN, one of algorithms showing the highest performance, Figure 7 shows that DTKNN significantly improves the performance for all metrics in the integrated feature.

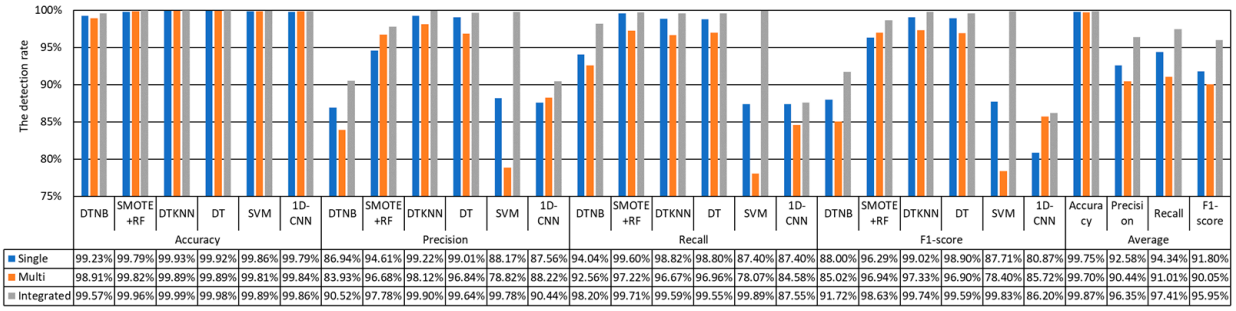
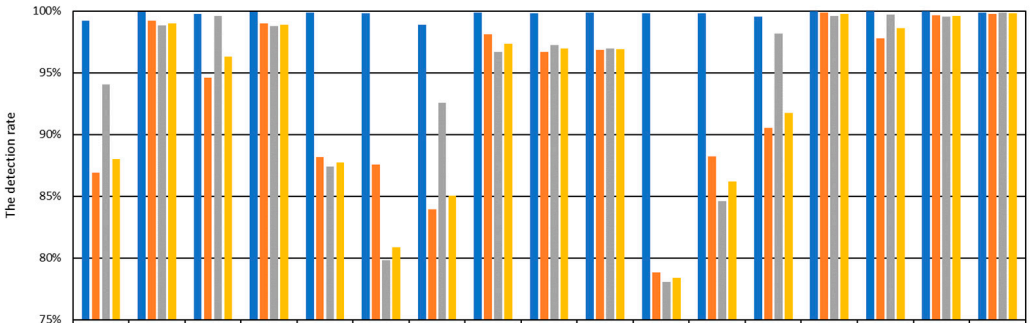


Figure 6. Detection rates in F1-score for machine learning models, according to each feature set.



	Single						Multi						Integrated				
	DTNB	DTKNN	SMOTE +RF	DT	SVM	1D-CNN	DTNB	DTKNN	SMOTE +RF	DT	SVM	1D-CNN	DTNB	DTKNN	SMOTE +RF	DT	SVM
Accuracy	99.23%	99.93%	99.79%	99.92%	99.86%	99.79%	98.91%	99.89%	99.82%	99.89%	99.81%	99.84%	99.57%	99.99%	99.96%	99.98%	99.89%
Precision	86.94%	99.22%	94.61%	99.01%	88.17%	87.56%	83.93%	98.12%	96.68%	96.84%	78.82%	88.22%	90.52%	99.90%	97.78%	99.64%	99.78%
Recall	94.04%	98.82%	99.60%	98.80%	87.40%	79.84%	92.56%	96.67%	97.22%	96.96%	78.07%	84.58%	98.20%	99.59%	99.71%	99.55%	99.89%
F1-score	88.00%	99.02%	96.29%	98.90%	87.71%	80.87%	85.02%	97.33%	96.94%	96.90%	78.40%	86.20%	91.72%	99.74%	98.63%	99.59%	99.83%

Figure 7. Performance metrics for machine learning models, according to each feature set.

Similar to other algorithms, SVM has the highest performance when integrated features are applied to it. The performance with single features is the lowest compared to the others, but the performance improvement is the greatest with integrated features. Finally, the detection performance of SVM improves most when integrated features are used compared to other algorithms.

It is important to compare confusion matrices to analyze the performance of each class in detail. However, due to the lack of space, we only show DTKNN's confusion matrix, which shows the highest classification accuracy, instead of confusion matrices for all classification algorithms. Tables 8–10 show the confusion matrix for DTKNN.

Table 8. Confusion matrix for DTKNN using the single-host feature.

	Benign	FTP-Patator	SSH-Patator	DoS Hulk	DoS Slowhttp	DoS Slowloris	Web Bruteforce	Bot	DDoS	Portscan
Benign	651,858	0	8	2	6	5	10	74	0	2
FTP-Patator	1	1580	0	0	0	0	0	0	0	0
SSH-Patator	1	0	1138	0	0	0	0	0	0	0
DoS Hulk	5	0	2	28,870	0	0	0	0	0	0
DoS Slowhttp	19	0	0	0	26,864	5	1	0	0	1
DoS Slowloris	4	0	0	0	6	20,948	3	0	0	0
Web Bruteforce	1	0	0	0	0	0	584	0	0	3
Bot	53	0	0	0	0	0	0	814	0	0
DDoS	0	0	0	0	0	0	0	0	38,218	0
Portscan	398	0	1	0	0	0	0	0	0	63,457

The results show the highest accuracy in the benign, DoS Slowhttp, Bot, and DDoS classes with the integrated feature, while other classes show almost the same accuracy as single-host or multi-host features. On the other hand, when using the single-host feature, DoS Hulk, DoS Slowloris, Web Bruteforce, and Portscan classes have the highest accuracy, but the difference from the results of the integrated feature is very marginal. In addition to the multi-host feature, it shows the highest performance in the SSH-Patator class, but it shows almost the same accuracy as the integrated feature. Eventually, there are classes that are accurately detectable when using single host or multi-host features, but the advantage of both features can be exploited by using the integrated feature, indicating that an NIDS with the integrated feature achieves the same or higher accuracy for all classes than the existing two features.

Table 9. Confusion matrix for DTKNN using the multi-host feature.

	Benign	**FTP-Patator**	**SSH-Patator**	**DoS Hulk**	**DoS Slowhttp**	**DoS Slowloris**	**Web Bruteforce**	**Bot**	**DDoS**	**Portscan**
Benign	652,143	0	2	54	42	93	151	21	6	34
FTP-Patator	1	1580	0	0	0	0	1	0	0	1
SSH-Patator	1	0	1147	0	0	0	0	0	0	1
DoS Hulk	23	0	0	28,665	13	40	15	0	6	1
DoS Slowhttp	33	0	0	10	26,820	0	4	0	6	0
DoS Slowloris	13	0	0	77	0	20,762	1	0	1	0
Web Bruteforce	74	0	0	0	0	0	426	0	0	0
Bot	19	0	0	0	0	0	0	867	0	0
DDoS	17	0	0	66	1	62	0	0	38,198	1
Portscan	16	0	0	0	0	1	0	0	1	63,425

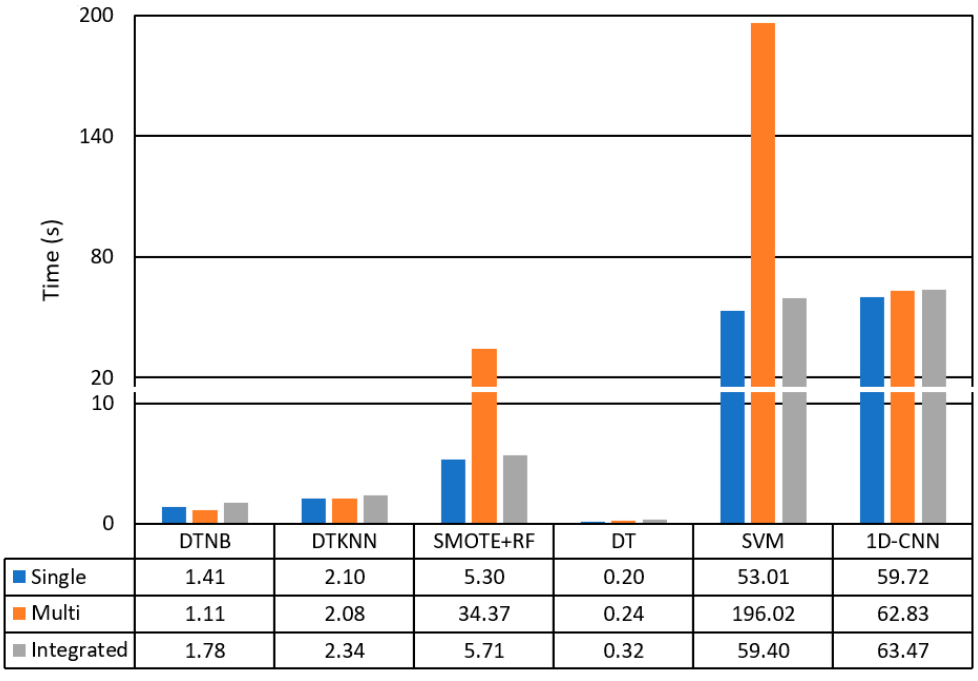
Table 10. Confusion matrix for DTKNN using the integrated feature.

	Benign	**FTP-Patator**	**SSH-Patator**	**DoS Hulk**	**DoS Slowhttp**	**DoS Slowloris**	**Web Bruteforce**	**Bot**	**DDoS**	**Portscan**
Benign	652,316	0	4	7	4	19	15	7	0	14
FTP-Patator	0	1580	0	0	0	0	0	0	0	0
SSH-Patator	0	0	1145	0	0	0	0	0	0	1
DoS Hulk	4	0	0	28,865	0	0	2	0	0	0
DoS Slowhttp	6	0	0	0	26,871	0	0	0	0	1
DoS Slowloris	3	0	0	0	1	20,939	0	0	0	0
Web Bruteforce	0	0	0	0	0	0	581	0	0	2
Bot	4	0	0	0	0	0	0	881	0	0
DDoS	1	0	0	0	0	0	0	0	38,218	0
Portscan	6	0	0	0	0	0	0	0	0	63,445

Single-host features reflect the details of a session between one source and one destination [25,26]. Multi-host features, on the other hand, contain aggregated information about sessions between multiple sources and a single destination. Thus, the information provided by single-host and multi-host features can reveal different levels of information for a specific session without duplication, allowing for a more detailed analysis of the session. Ultimately, this synergy leads to an integrated feature-based NIDS outperforming single-host feature or multi-host feature-based NIDSs in detection.

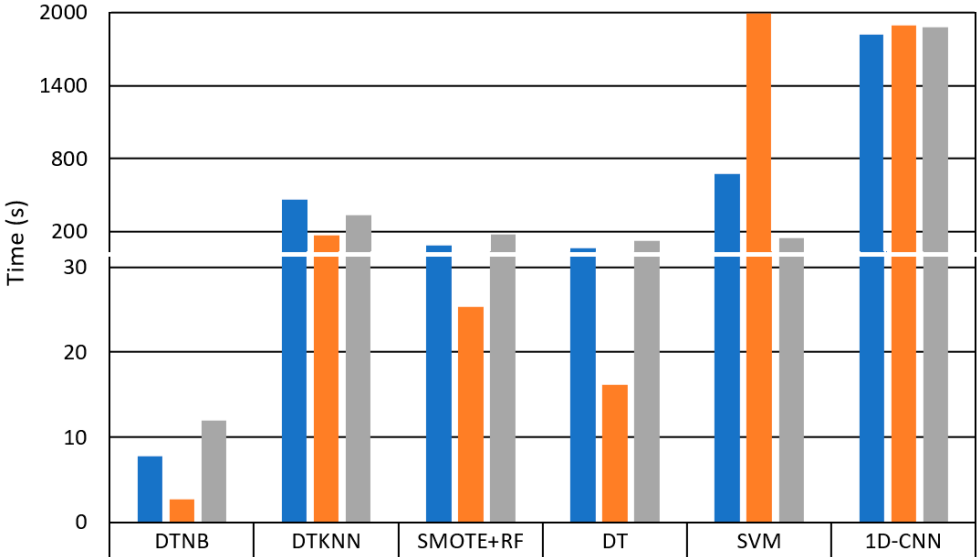
4.3. Training and Testing Time

As indicated by the training time results in Figure 8, the ML model using the multi-host feature has the shortest training time. In contrast, the NIDS using integrated features demonstrates the longest training time. Considering that the size of the multi-host feature is the smallest and the size of the integrated feature is the largest, we can realize that the training time tends to be proportional to the number of features. As the multi-host feature size is the smallest, the training time is the smallest compared to the use of other features for all algorithms. An NIDS using the multi-host feature can be trained almost three times faster than the single-host feature. Using the integrated feature with the largest size usually requires more training time than the single-host feature.



	DTNB	DTKNN	SMOTE+RF	DT	SVM	1D-CNN
Single	1.41	2.10	5.30	0.20	53.01	59.72
Multi	1.11	2.08	34.37	0.24	196.02	62.83
Integrated	1.78	2.34	5.71	0.32	59.40	63.47

(**a**)



	DTNB	DTKNN	SMOTE+RF	DT	SVM	1D-CNN
Single	7.82	461.20	83.77	67.68	676.00	1818.72
Multi	2.76	173.76	25.29	16.19	1995.00	1894.23
Integrated	11.97	332.55	177.17	123.16	150.72	1882.81

(**b**)

Figure 8. Each algorithm's relative training and testing time according to each feature set. (**a**) Training time. (**b**) Testing time.

One of the most critical aspects of NIDS is the testing time. This is an important performance factor because it determines the maximum processing throughput an NIDS is capable of handling. As shown in Figure 8, the characteristics of the primary testing time are similar to those of the training time. Similar to the training time results excluding DTKNN, the NIDS using the multi-host feature shows the shortest testing time, whereas the NIDS using the integrated feature has the longest testing time. However, unlike the training time result, the testing time of the NIDS using integrated features does not increase significantly compared to the case of an NIDS using single-host features. In the case of the

DTKNN with the highest detection rate, the testing time only increases by 8%. Even for the most significant increase in DT, the testing time only increases by 61%. Although the test speed is lower using the integrated features, the performance is almost similar to that of the multi-host feature.

For SVMs, both the training and test times are the shortest for integrated features and the longest for multi-host features. Considering that it has the fewest number of multi-host features and the largest number of integrated features, the results for SVM seem very strange and hard to understand. However, if we note that the time complexity of the SVM is between $O(n^2)$ and $O(n^3)$, depending on the data distribution, we can see that the number of features affects the time complexity of the classifiers, but the distribution of the samples has a greater impact on the time complexity of SVM [27,28].

4.4. Feature Selection

To resolve the problem of increasing the classification time when using integrated features, it is necessary to compare and analyze the performance through feature selection [29]. In this experiment, a random forest is used for feature selection, the feature importance is calculated, and k features with high importance values are selected [30]. The classifier is then trained using only the selected features, the F1-score and test time are measured. Since most classifiers show similar results, we show here the results only for DT, which shows the largest increase in test time when using integrated features.

Table 11 shows the F1-score and test time according to the size of the features selected through feature selection. We can see that as the number of selected features decreases, the test time decreases proportionally. This is because as the number of features increases, the complexity of building the internal tree of DT increases. On the other hand, the F1-score shows an overall convex shape with respect to the number of features. The table shows that the F1-score is maximized when the number of features is 30 and the test time is decreased to 0.145 s, which is 54.7% less than the original 0.32 s. Considering that the numbers of single-host and multi-host features are 81 and 39, the DT based on integrated features using 39 features shows a shorter test time than those of a DT using single-host or multi-host features. Therefore, it is confirmed that the increased test time caused by using integrated features can be mitigated by feature selection. Moreover, feature selection is essential for using integrated features because feature selection improves not only test time but also the F1-score.

Table 11. Comparison of F1-score and test time for integrated feature-based DT, according to the size of features selected by feature selection.

Feature Size	F1–Score	Test Time (s)
116	99.59%	0.320
81	99.60%	0.246
39	99.78%	0.148
34	99.81%	0.138
30	99.88%	0.145
27	99.59%	0.126
25	99.52%	0.124
20	99.45%	0.128

5. Conclusions

The existing single-host feature is easy to create but has a disadvantage—it is unable to see relationships with other sessions. However, the multi-host feature is difficult to create but has the advantage of knowing the relationship with other sessions. Therefore, in this study, we proposed a method to integrate the two sessions. Since generating integrated features has a significant overhead, we also proposed an incremental feature generation

method to solve this problem. The proposed integrated-session feature has the strength of being able to more accurately detect network intrusions by merging multi- and single-host features to simultaneously use multi- and single-session information. The experimental results indicate that the proposed integrated feature improves the detection rate by 4.15% and 5.9% on average, respectively, compared to NIDSs using traditional single-host and multi-host features.

As the number of features increases, the time required to classify the received session through ML increases. More powerful hardware is required to support the same session-processing speed. However, an NIDS based on integrated session features can improve the classification performance by adopting feature selection or using multiple ML classifiers in parallel. In addition, because the performance of hardware for ML is rapidly improving, the slow classification speed will be technically overcome.

For more accurate network intrusion detection, feature set design is essential; however, research on this has been relatively lacking. Therefore, the integrated session feature proposed in this study could significantly help to design a feature set for NIDSs that can safely protect networks and users from malicious users.

Author Contributions: T.K. and W.P. have written this paper and have conducted the research. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant, funded by the Korean government (Ministry of Science and ICT) (NRF-2022R1A2C1011774).

Data Availability Statement: The dataset utilized in this paper is CIC-IDS2017 dataset (https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 6 March 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), Shah Alam, Selangor, Malaysia, 17–18 May 2010; pp. 200–203. [CrossRef]
4. 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]
5. 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]
6. Al-Qatf, M.; Lasheng, Y.; Al-Habib, M.; Al-Sabahi, K. Deep Learning Approach Combining Sparse Autoencoder with SVM for Network Intrusion Detection. *IEEE Access* **2018**, *6*, 52843–52856. [CrossRef]
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 December 2009. [CrossRef]
9. Shiravi, A.; Shiravi, H.; Tavallaee, M.; Ali, A. Ghorbani, 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), Funchal-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. 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]

13. Lashkari, A.H.; Draper-Gil, G.; Mamun, M.; Ghorbani, A.A. Characterization of Tor Traffic Using Time Based Features. In Proceeding of the 2017 3rd International Conference on Information System Security and Privacy, Porto, Portugal, 19–21 February 2017; SCITEPRESS: Setúbal, Portugal. [CrossRef]
14. 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]
15. Ma, C.; Du, X.; Cao, L. Analysis of Multi-Types of Flow Features Based on Hybrid Neural Network for Improving Network Anomaly Detection. *IEEE Access* **2019**, *7*, 148363–148380. [CrossRef]
16. Panwar, S.S.; Raiwani, Y.P.; Panwar, L.S. An Intrusion Detection Model for CICIDS-2017 Dataset Using Machine Learning Algorithms. In Proceedings of the International Conference on Advances in Computing, Communication and Materials (ICACCM), Dehradun, India, 10–11 November 2022; pp. 1–10. [CrossRef]
17. Uhm, Y.; Pak, W. Real-Time Network Intrusion Prevention System Using Incremental Feature Generation. *CMC-Comput. Mater. Contin.* **2022**, *70*, 1631–1648. [CrossRef]
18. 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]
19. Description of Kyoto University Benchmark Data. Available online: https://www.takakura.com/Kyoto_data/BenchmarkData-Description-v5.pdf (accessed on 13 January 2023).
20. Han, X.; Dong, P.; Liu, S.; Jiang, B.; Lu, Z.; Cui, Z. IV-IDM: Reliable Intrusion Detection Method based on Involution and Voting. In Proceedings of the 2022 IEEE International Conference on Communications (ICC), Seoul, Republic of Korea, 16–20 May 2022; pp. 4162–4167. [CrossRef]
21. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; Kegelmeyer, W.P. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357. [CrossRef]
22. Yan, B.; Han, G.; Sun, M.; Ye, S. A novel region adaptive SMOTE algorithm for intrusion detection on imbalanced problem. In Proceedings of the 2017 IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 13–16 December 2017; pp. 1281–1286. [CrossRef]
23. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
24. Kiranyaz, S.; Avci, O.; Abdeljaber, O.; Ince, T.; Gabbouj, M.; Inman, D.J. 1D convolutional neural networks and applications: A survey. *Mech. Syst. Signal Process.* **2021**, *151*, 107398. [CrossRef]
25. Wang, W.; Harrou, F.; Bouyeddou, B.; Senouci, S.-M.; Sun, Y. Cyber-attacks detection in industrial systems using artificial intelligence-driven methods. *Int. J. Crit. Infrastruct. Prot.* **2022**, *38*, 100542. [CrossRef]
26. Dairi, A.; Harrou, F.; Bouyeddou, B.; Senouci, S.M.; Sun, Y. Semi-supervised Deep Learning-Driven Anomaly Detection Schemes for Cyber-Attack Detection in Smart Grids. In *Power Systems Cybersecurity. Power Systems*; Springer: Cham, Switzerland, 2023; pp. 265–295. [CrossRef]
27. Bottou, L. Support Vector Machine Solvers. Available online: https://leon.bottou.org/publications/pdf/lin-2006.pdf (accessed on 23 March 2023).
28. Simon, H.; List, N. SVM-Optimization and Steepest-Descent Line Search. In Proceedings of the 22nd Conference on Learning Theory (COLT), Montreal, QC, Canada, 18–21 June 2009.
29. Guyon, I.; Elisseeff, A. An Introduction to Variable and Feature Selection. *JMLR* **2003**, *3*, 1157–1182.
30. Ho, T.K. The Random Subspace Method for Constructing Decision Forests. *IEEE Trans. Pattern Anal. Mach. Intell.* **1998**, *20*, 832–844. [CrossRef]