Hindawi
Security and Communication Networks
Volume 2023, Article ID 9790342, 1 page
https://doi.org/10.1155/2023/9790342

Retraction

Retracted: Detecting Temporal Attacks: An Intrusion Detection System for Train Communication Ethernet Based on Dynamic Temporal Convolutional Network

Security and Communication Networks

Received 26 December 2023; Accepted 26 December 2023; Published 29 December 2023

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] C. Yue, L. Wang, D. Wang, R. Duo, and H. Yan, "Detecting Temporal Attacks: An Intrusion Detection System for Train Communication Ethernet Based on Dynamic Temporal Convolutional Network," *Security and Communication Networks*, vol. 2021, Article ID 3913515, 21 pages, 2021.

Hindawi
Security and Communication Networks
Volume 2021, Article ID 3913515, 21 pages
https://doi.org/10.1155/2021/3913515

Research Article

Detecting Temporal Attacks: An Intrusion Detection System for Train Communication Ethernet Based on Dynamic Temporal Convolutional Network

Chuan Yue, Lide Wang, Dengrui Wang, Ruifeng Duo, and Haipeng Yan

School of Electrical Engineering, Beijing Jiaotong University, Beijing 100044, China

Correspondence should be addressed to Chuan Yue; 16117394@bjtu.edu.cn

Received 31 May 2021; Revised 22 July 2021; Accepted 6 August 2021; Published 18 August 2021

Academic Editor: Chi-Hua Chen

The train communication Ethernet (TCE) of modern intelligent trains is under an ever-increasing threat of serious network attacks. Denial of service (DoS) and man in the middle (MITM), the two most destructive attacks against TCE, are difficult to detect by conventional methods. Aiming at their highly time-correlated properties, a novel dynamic temporal convolutional network-based intrusion detection system (DyTCN-IDS) is proposed in this paper to detect these temporal attacks. A semi-physical TCE testbed that is capable of simulating real situations in TCE-based trains is first built to generate an effective dataset for training and testing. DyTCN-IDS consists of two phases, and in the first phase, systematic feature engineering is designed to optimize the dataset. In the second phase, a basic detection model that is good at dealing with temporal features is first built by utilizing the temporal convolutional network with several architectural optimizations. Then, in order to decrease the computational consumption waste on network packet sequences with different lengths of inner temporal relationships, dynamic neural network technology is further adopted to optimize the basic detection model. Diverse experiments were carried out to evaluate the proposed system from different angles. The experimental results indicate that our system is easy to train, converges fast, costs less computational resources, and achieves satisfying detection performance with a macro false alarm rate of 0.09%, a macro F-score of 99.39%, and an accuracy of 99.40%. Compared to some canonical DL-based IDS and some latest IDS, our system acquires the best overall detection performance as well.

1. Introduction

Train communication network is a typical industrial control system in the railway domain, which enables information exchange throughout a train [1]. There is much more smart equipment, e.g., intelligent sensors and smart control units [2, 3], attached to the train communication network, with the rapid development of intelligence in rail vehicles. The traditional train communication network, composed of wire train bus and multifunction vehicle bus, cannot meet the demands of growing bandwidth and interconnection. Therefore, the Ethernet technology with the merits of large bandwidth and strong flexibility is emerging as the new major architecture for the train communication network, which is also known as the train communication Ethernet (TCE).

Although TCE improves data communication within the train, it is associated with a large number of cybersecurity issues. In a recent whitepaper, UNIFE, the association representing European rail supply companies, identified the cybersecurity as one of the five most critical focus areas for rail [4]. Unlike commercial scenarios, trains are closely related to the safety of passengers. Almost all information is transmitted through TCE, and this includes critical control instructions (e.g., traction instructions and brake instructions). Unfortunately, malicious network intrusions can disrupt normal communications, which could lead to unexpected breakdowns of a train, or even loss of lives. Among all kinds of network attacks, denial of service (DoS) attacks and man in the middle (MITM) attacks were extremely destructive to TCE. DoS attacks terminate communications by exhausting the computing resources of network

equipment, which breaches the availability of TCE. MITM attacks have the capability of replaying or tampering crucial control messages in TCE to cause serious accidents. For example, a brake instruction could be replayed by a MITM attack after a certain time, resulting in the brake control unit unable to brake timely. Furthermore, this could lead to a severe brake incident. One thing in common with these two kinds of network attacks is that they disorganize the original temporal order of normal communications in TCE. In other words, network packets of these attacks appear in an inappropriate time place to destroy availability and integrity of TCE. We often refer to them as temporal attacks because of their highly time-correlated properties [5]. Since TCE is an infrastructure network system with high real-time and reliability requirements, these temporal attacks are definitely intolerable in TCE. Therefore, it is vital to take necessary measures to counter temporal attacks targeted at TCE.

The intrusion detection system (IDS) is a defense system that provides a wall of defense, which overcomes cyberattacks, and it is based on an assumption that the behaviors of intruders differ from legal users [6]. IDS performs the critical task of detecting malicious network attacks, and the detection results of IDS could be utilized as alerting signals to network operators. IDS can be considered as the concrete implementation and deployment of intrusion detection methods in network systems. Broadly, according to the detection method, IDS could be categorized into two classes, namely, signature-based IDS and anomaly-based IDS [7]. Signature-based IDS refers to detecting attacks by searching for specific patterns, such as byte sequences in network packets. An obvious shortcoming of signature-based IDS is that it is incapable of detecting unknown attacks, for which no signature pattern is available. On the contrary, anomaly-based IDS detects attacks by identifying deviations between observed packets and normal packets, enabling to find out unseen attacks.

The huge volume of network data has made intrusion detection issues amenable to machine learning (ML) methods, which have been successfully applied in natural language processing [8], recommendation system [9], etc. ML-based IDS has attracted much interest from researchers over the last decade [10–12]. ML-based IDS could be roughly divided into pattern recognition issues and anomaly detection issues despite its fuzzy boundary, which means that it overlaps with anomaly-based IDS to a large extent. Hence, researchers sometimes attribute ML-based IDS to anomaly-based IDS. In general, ML-based intrusion detection is a classification problem in nature, classifying the network data into normal class and various attack classes. ML methods can be broadly grouped into two categories: shallow learning methods (e.g., support vector machine, K-nearest neighbor, and decision trees) and deep learning (DL) methods (e.g., convolutional neural network (CNN), recurrent neural network (RNN), and temporal convolutional network (TCN)). The application of DL methods in IDS has achieved excellent performance due to IDS's strong fitting ability of nonlinear functions [13–15].

Because the spatial features of these temporal attacks are extremely similar to those of benign packets in TCE, which means that traditional spatial features do not have enough contribution to intrusion detection algorithms, according to the above introduction of these temporal attacks, we choose to utilize their temporal relationships between continuous network packets to be the main learning features for our intrusion detection algorithm. In this paper, utilizing the advantages of TCN, while improving its network structure by some architectural optimization and dynamic neural network technologies, we propose an IDS named DyTCN-IDS dedicated to detecting underlying temporal attacks against TCE. On a TCE testbed built by us, we demonstrate that our IDS reaches a prominent performance both in detection accuracy and computational resources consumption. To the best of our knowledge, no researchers have conducted systematic intrusion detection research aiming at TCE temporal attacks till date, and this paper is the first attempt to deal with this issue. The specific contributions of this paper can be summarized as follows:

(1) Specific underlying temporal attacks against TCE are analyzed, and an effective dataset is generated from a self-built semiphysical TCE testbed.

(2) A systematic feature engineering is designed to optimize our dataset. The temporal convolutional network with several architectural optimizations is utilized to build a basic TCN architecture for intrusion detection. Moreover, dynamic neural network technology is adopted to optimize our basic TCN and finally construct our DyTCN-IDS.

(3) Various experiments were conducted to evaluate the performance of the proposed system, and the detailed analytical processes are given. Experimental results indicate that the proposed DyTCN-IDS can detect temporal attacks against TCE accurately and efficiently.

The remainder of this paper is organized as follows. The background of TCE and the description of our testbed are given in Section 2. The proposed intrusion detection system is described in detail in Section 3. In Section 4, experiment operations are represented, and the experimental results are analyzed. Finally, Section 5 highlights the conclusions of this paper.

2. Background and Testbed

In this section, we first give a literature survey of relevant works. For a better understanding of our work, the basic background of TCE and the description of our testbed are then given. The network topology of TCE is described. Afterwards, the main protocols of TCE are given, based on which the underlying temporal intrusions are analyzed. In order to create an appropriate evaluation environment, a semiphysical testbed dedicated to TCE is built. Finally, details of the testbed including equipment compositions and communication configurations are given.

2.1. Literature Survey. Extensive literature reviews on shallow ML-based or DL-based IDS have been published in recent years. They can be broadly divided into two parts. The

first part is works that do not concern temporal relationships. Vijayakumar and Ganapathy [16] made an extensive survey about the role of ML techniques to reduce the false alarm rate in WIDS. They proved that substantial improvements have been achieved by reducing false alarm rate by ML algorithms. In addition, they also proposed a filtration technique that has the capability of enhancing the performance in terms of reduction of false alarm rate. Man and Sun [17] proposed a residual learning-based intrusion detection model. They utilized a modified focal loss function to deal with the class imbalance problem existing in the UNSW-NB15 dataset. Experimental results show that their model can improve detection accuracy compared with several existing models. Riyaz and Ganapathy [18] proposed an IDS that is capable of identifying and detecting the intruders effectively in wireless networks. They proposed a conditional random field and linear correlation coefficient-based feature selection algorithm to select the most contributed features and then classify them using CNN. Experimental results indicate that their IDS achieves 98.88% as overall detection accuracy. Xu et al. [19] designed an IDS on the basis of a meta-learning method to solve the few-shot detection problem. Their method achieves an outstanding average detection rate on their self-built dataset. Li et al. [20] designed DAS-CIDS by applying disagreement-based semisupervised learning algorithm for collaborative IDS. Both KDD99 dataset and a dataset collected in a real IoT environment are used to evaluate their method. Experimental results indicate that their approach is more effective in detecting intrusions and reducing false alarms by automatically leveraging unlabeled data when compared with traditional supervised classifiers.

The second part is works that concern temporal relationships. Nancy et al. [21] proposed an IDS that enables the detection of known and unknown types of attacks using an intelligent decision tree algorithm, where a dynamic recursive feature selection algorithm and an intelligent fuzzy temporal decision tree algorithm are designed. Experiments were carried out using KDD cup dataset and network trace dataset. The results indicate that the false positive rate, energy consumption, and delay are reduced by their method. Hsu et al. [22] proposed an IDS method based on convolutional neural network (CNN) and long short-term memory (LSTM). Using NSL-KDD dataset as the benchmark, they evaluated their method, and the results indicate that the method outperforms several existing works. Gao et al. [23] proposed an omni-IDS for supervisory control and data acquisition networks. They combined LSTM and FNN through an ensemble approach and validated their method on both temporally uncorrelated attack data and temporally correlated attack data. The results show that the method achieves a promising F1 score. Wang et al. [24] put forward a hierarchical spatial-temporal feature-based intrusion detection system, which first learns the low-level spatial features of network traffic using CNN and then learns high-level temporal features using LSTM. They use standard DARPA 1998 and ISCX2012 dataset to evaluate the performance, and the results indicate that their system works well.

The above works obtained promising performance in various intrusion detection scenarios; however, there are still some limitations and problems left to be addressed. First, previous works are mainly evaluated upon several classical public network intrusion datasets. The network scenarios reflected by these datasets have obvious differences with TCE, which means that these works may not work well with attacks against TCE. Second, most former works are targeted for multitudinous attacks. They do not necessarily perform well on temporal attacks due to the lack of specialized considerations on temporal correlation within network packets. Third, although there are some works employing RNN methods (e.g., LSTM and GRU) that have the capability to deal with temporal issues, yet these methods are difficult to train [25] and cost high detection time [13]. Some works using shallow ML algorithms concerning temporal relationships may not work well on very long temporal sequences.

In order to detect underlying temporal attacks against TCE, while breaking through the limitations of existing methods, this paper proposes a DyTCN-IDS, which is described in detail in Section 3.

2.2. Network Topology of TCE. Since 2014, the International Electrotechnical Commission (IEC) has formally brought industrial Ethernet into the train communication network through its IEC61375 series standards, thus forming the TCE network system. Defined in IEC61375-2-3 [26], the conventional network topology of TCE consists of two segments, namely, the Ethernet train backbone (ETB) and the Ethernet consist network (ECN), as shown in Figure 1. ETB is a train-wide communication network that enables data transmission among vehicles within one train (usually eight or sixteen vehicles per train) via physical lines connecting the train-level active network devices together (ETBN, Repeater, etc.) [27]. ECN is a vehicle-wide communication network that enables data transmission among end devices within one vehicle via the physical lines connecting vehicle-level active network devices (CS, Repeater, etc.) and all kinds of end devices [28].

2.3. Main Protocols of TCE. TCE adopts a specially designed train real-time data protocol (TRDP) for high-level real-time communication [26]. TRDP is designed on the basis of Ethernet/IP stack and located between the application layer and the transport layer, as illustrated in Figure 2. It enables the exchange of process data (PD) and message data (MD) between end devices (e.g., master controller, door control units, video units, and event recorders) based on user datagram protocol (UDP) and optionally on transmission control protocol (TCP). A packet transmitted in TCE has a TRDP header added before the transport layer header is added. Address resolution protocol (ARP) is utilized to discover physical addresses associated with known IP addresses. Additionally, the Internet control message protocol (ICMP) and Internet group management protocol (IGMP) are employed for network management.

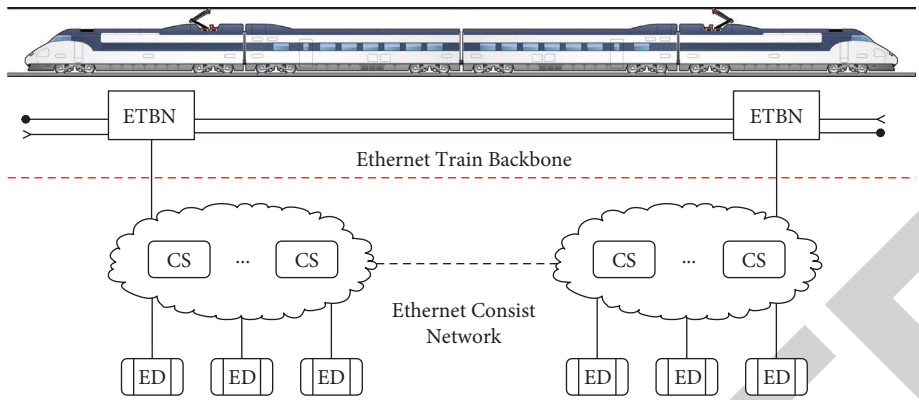

Figure 1: Conventional network topology of TCE. ETBN stands for Ethernet Train Backbone Node, CS stands for Consist Switch, and ED stands for End Device.

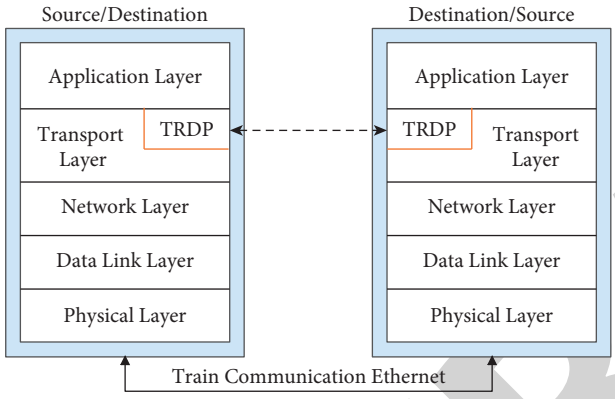

Figure 2: Protocol stack of TCE. TRDP is located between the application layer and the transport layer.

2.4. Underlying Temporal Intrusions against TCE. As mentioned in Section 1, DoS attacks and MITM attacks are two main types of underlying temporal intrusions against TCE. A DoS attack is an intrusion meant to slow down the data transmission or even shut down the network, making the network service unavailable. It accomplishes this by flooding the network with a massive amount of network packets or sending unexpected packets as a blasting fuse to trigger a network crash [29]. Although DoS attacks come up a lot in commercial networks, people do not pay much attention to them. That is because it usually just makes sufferers unable to enjoy their entertainment time without doing too much damage to personnel's property. However, the consequence is opposite when it comes to TCE. A DoS attack intruded in TCE, where the most transmitted information is critical control message, will make control instructions unavailable. This will probably lead to an unexpected operation accident. According to the weaknesses of protocols within TCE, several categories of DoS attacks could happen: UDP flood attacks [30] exploiting the weaknesses of TRDP and UDP, SYN flood attacks [31] exploiting the weaknesses of TRDP and TCP, MAC flood attacks [32] exploiting the weaknesses of switch tables, and Smurf attacks [33] exploiting the weaknesses of ICMP.

A MITM attack is an intrusion where the intruder hides itself in a middle position between two devices to tamper or replay the normal messages, making it appear as if the normal exchange of network packets is ongoing normally. Assuming that there are two end devices in TCE exchanging information (one of them is a master controller sending deceleration instructions from the train driver, and the other is the traction and brake control unit), a MITM attack can intercept the instructions and then replay them after a long delay or directly tamper them to acceleration instructions. It is easy to imagine that it is about to cause a train operation incident and pose a threat to the safety of passengers. There are three kinds of MITM attacks that are highly likely to occur in TCE: Tamper attacks [34], Replay attacks [35], and ARP spoofing attacks [36].

There are obvious temporal correlations existing in both DoS attacks and MITM attacks as described in Section 1. DoS attacks can cause a huge number of network packets, which disrupts the original time sequence of normal communication (normal communication of TCE usually occurs periodically). This makes DoS attacks intruded in TCE have temporal characteristics. MITM attacks are generally difficult to detect, because they have almost the same features with normal packets. Fortunately, as the saying goes, all that pass by leave a trace, and MITM is no exception. A MITM attack, either tamper or replay, is bound to consume time, which means that the tampered or replayed packets appear in new positions. The change in position of normal and MITM packets results in a unique temporal correlation. In our proposed method, in Section 3, we make full use of the temporal characteristics of DoS attacks and MITM attacks to detect them.

2.5. TCE Testbed. Because of the data privacy limitations of the railway industry, there is no available public dataset for the evaluation of intrusion detection issues of TCE. To overcome this problem, we built a semiphysical TCE testbed based on a certain type of train (following the general TCE topology described in Section 2.2), which has been put into public use in America (the name of the train could not be

given because of the confidentiality requirements). The implemented TCE testbed is shown in Figure 3.

Our testbed comprises the following devices: (1) two TP-link L3 Ethernet switches (TL-SH6428) as the ETBNs, (2) four TP-link L3 Ethernet switches (TL-SG5218) as the CSs, (3) twelve embedded Linux development boards as the end devices, (4) one ALINX development board (ZYNQ7000) as the intrusion device, and (5) one PC as the data acquisition device.

TRDP stack is ported into every end device, and the normal train communication traffics are configured, referencing the actual train communication services configurations. An intrusion software that is capable of injecting DoS and MITM attacks is developed by Python. Network data can be captured by the data acquisition device, utilizing the switched port analyzer technology and the Wireshark software. To be specific, except for benign packets, four types of DoS attacks, namely, UDP flood, SYN flood, MAC flood, and Smurf, and three types of MITM attacks, namely, Tamper, Replay, and ARP spoofing, are implemented, respectively.

3. Proposed DyTCN Intrusion Detection System

The proposed DyTCN intrusion detection system has two main components: feature engineering and detection method, as illustrated in Figure 4. The feature engineering can create a suitable data environment for our detection method. The detection method essentially is a classification method and is used to detect temporal network intrusions. Detailed descriptions about the whole detection system are provided in the following subsections. The data collection part is a preportion of our system, so for the convenience of reading, we give a description of it in this section first.

3.1. Data Collection: TCE-IDS Dataset. To build an experimental dataset that represents TCE network traffic, we first generate benign background traffic in our testbed, according to the network configuration files of the real train mentioned in Section 2.5. Furthermore, four types of DoS attacks (namely, UDP flood, SYN flood, MAC flood, and Smurf) and three types of MITM attacks (namely, Tamper, Replay, and ARP spoofing) are implemented in our testbed. Finally, using the Wireshark software to capture the raw packets, we build a realistic dataset including normal packets (labelled as benign) and seven types of attacks (labelled as their type names). For description convenience, our dataset is named as the TCE-IDS dataset. The total number of instances in the TCE-IDS dataset is 944860, and the data distribution of the dataset is illustrated in Figure 5.

3.2. Feature Engineering. By conducting real network attacks in our testbed, we can obtain a dataset that contains normal packets and malicious packets. In order to establish a suitable data environment for our detection method, all the network data should be preprocessed appropriately (i.e., applying feature engineering). The feature engineering phase of our system consists of feature extraction, handling missing values, data transformation, label transformation, and temporal sequence construction. These five parts are executed sequentially; namely, each part is triggered after the previous part is finished.

3.2.1. Feature Extraction and Selection. Raw network packets have irregular length, in which the data is noisy and represented by the format of hexadecimal character. It is meaningful to extract effective features from the raw network packets to form a more efficient data representation. Firstly, network packets ware captured using Wireshark software and saved in a pcap file. From this file, all the header features of TCE network packets are extracted to be columns in a csv file. Network packets are constructed starting from the application layer and encapsulating the corresponding protocol headers layer by layer. From the headers of several major protocols within TCE network packets (i.e., IP, UDP, TCP, ICMP, TRDP, etc.), we select valid features sufficiently to cover as much information as possible. The 32 selected features with acronyms in brackets are listed in Table 1, where the descriptions of data type are also given. Feature 1 is the protocol type of the packet, and feature 2 is the total length of the packet. These two features are basic features shared by all the packets. Features 3–9 are information of IP headers. Because all the TCE packets are based on IP, these features are also globally shared. Features 10–12 are information of UDP headers of UDP-based TRDP packets, and features 13–19 are information of TCP headers of TCP-based TRDP packets. Features 20–22 are information of ICMP headers. Features 23-24 are source and destination MAC address of TCE packet. Features 25–32 are features of TRDP headers specific to TCE. We use this feature extraction and selection manner to avoid information omission as much as possible.

3.2.2. Handling Missing Values. Differences between protocol types may result in certain features of some instances being unavailable, which means that there are some missing values in our dataset. For instance, the features extracted from ICMP are only effective in the ICMP packets; that is to say, the features Type_ICMP, Code_ICMP, and ID_ICMP in other types of packets do not have values. This kind of missing data belongs to the class of Missing not at Random (MNAR) [37], so a missing value imputation technology is needed. Zero imputation [38] that has the merit of being capable of reducing the influence of subjective factors on the system is adopted in this paper.

3.2.3. Data Transformation. There are three types of features as can be seen in Table 1, namely, nominal type, string type, and numerical type. Nominal features and string features cannot be fed into neural network models; thus, we use one-hot encoding to transform nominal features into numerical types. It is worth noting that one-hot encoding is not suitable for string features like Src_IP and Dst_IP, because these features have too many values, and the usage of one-hot encoding will lead to a curse of dimensionality [39]. Ordinal

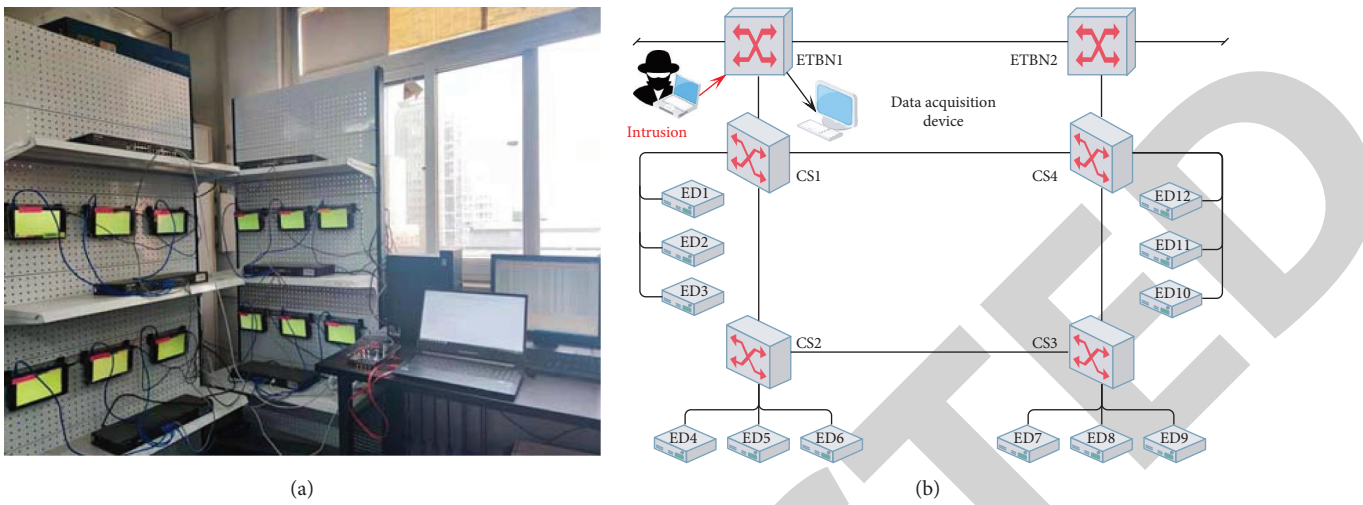

Figure 3: Implemented TCE testbed. (a) The physical photograph of the testbed. (b) The logistic topology of the testbed. Every node and line in (a) and (b) have one-to-one correspondence.

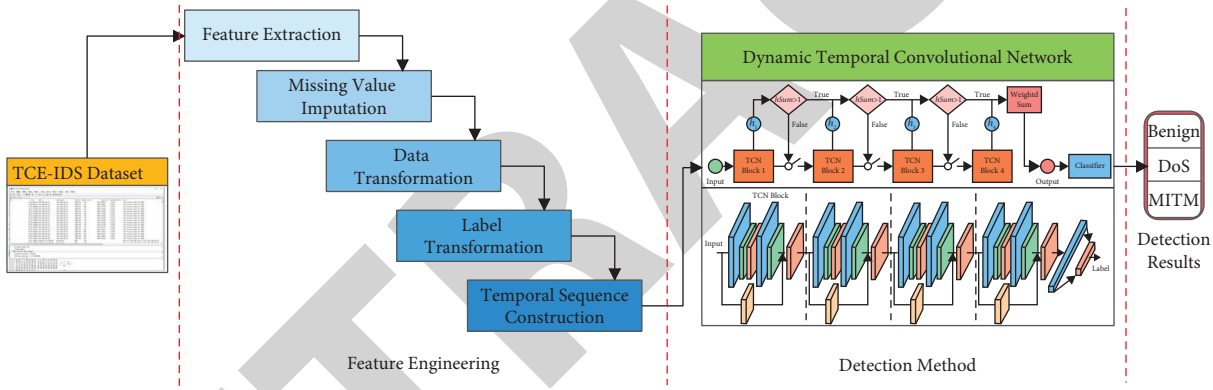

Figure 4: Framework of the proposed DyTCN intrusion detection system.

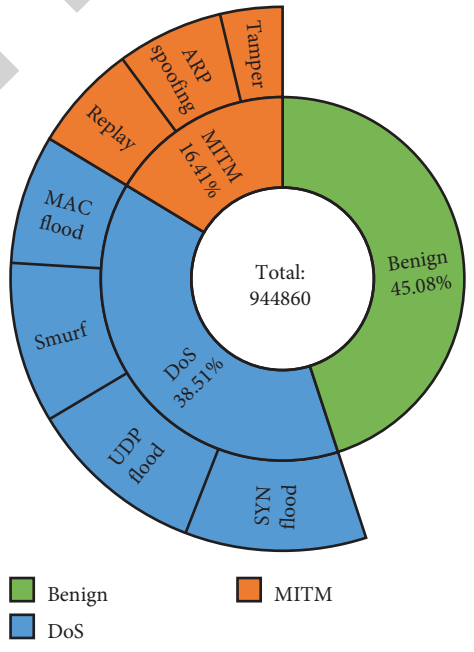

Figure 5: Data distribution of the TCE-IDS dataset.

Table 1: Extracted features of the TCE-IDS dataset.

No.	Description	Type
1	Protocol type (protocol)	Nominal
2	Total length (Len)	Numerical
3	Source IP address (Src_IP)	String
4	Destination IP address (Dst_IP)	String
5	Time to live of IP (TTL_IP)	Numerical
6	Differentiate services field (DSF_IP)	Numerical
7	Length of IP (Len_IP)	Numerical
8	Identification of IP (ID_IP)	Numerical
9	Flag of IP (Flag_IP)	Numerical
10	UDP source port (SrcPort_UDP)	Numerical
11	UDP destination port (DstPort_UDP)	Numerical
12	Length of UDP (Len_UDP)	Numerical
13	TCP source port (SrcPort_TCP)	Numerical
14	TCP destination port (DstPort_TCP)	Numerical
15	Length of TCP (Len_TCP)	Numerical
16	Sequence number of TCP (Seq_TCP)	Numerical
17	Acknowledge number of TCP (Ack_TCP)	Numerical
18	Flag of TCP (Flag_TCP)	Nominal
19	Window size value (WinSize)	Numerical
20	Type of ICMP (Type_ICMP)	Numerical
21	Code of ICMP (Code_ICMP)	Numerical
22	Identifier of ICMP (ID_ICMP)	Numerical
23	Source MAC address (Src_MAC)	String
24	Destination MAC address (Dst_MAC)	String
25	TRDP sequence number (Seq_TRDP)	Numerical
26	TRDP message type (MsgType_TRDP)	Nominal
27	TRDP communication ID (ComId_TRDP)	Numerical
28	ETB topology counter (Etc_TRDP)	Numerical
29	Operation topology counter (Otc_TRDP)	Numerical
30	Length of TRDP (Len_TRDP)	Numerical
31	Reply ComID (ReComId_TRDP)	Numerical
32	Reply IP (ReIP_TRDP)	String

encoding is adopted in our IDS to convert string features to numerical types.

To balance the comparability of features with a large initial range and features with a small initial range, we utilize z-score normalization to normalize all the features. It is calculated as follows:

$$x' = \frac{x - \mu}{\sigma}, \tag{1}$$

where x denotes unnormalized data, x' denotes normalized data, and μ and σ are the mean value and the standard deviation of an original feature.

3.2.4. Label Transformation. The original labels of our dataset comprise eight types as described in Section 3.1. Since our intrusion detection issue could be formulated as a 3-class classification problem, UDP flood, SYN flood, MAC flood, and Smurf are mapped to DoS class; meanwhile, Tamper, Replay, and ARP spoofing are mapped to MITM class. The mapping relationship is shown in Table 2.

3.2.5. Temporal Sequence Construction. Our DyTCN-IDS receives temporal sequence data as input. The sequence of our dataset is formed by grouping a series of consecutive network packets, and the label of the sequence is denoted as the label of the last packet within the sequence, which means that the temporal information between a packet and former packets is aggregated in the sequence. The construction process of the temporal sequence is illustrated in Figure 6.

All the sequences have the same length seq Len, which determines the maximum amount of temporal information carried in a sequence and acts as a hyperparameter in our system.

Table 2: Mappings of original labels to transformed labels.

Original labels	Transformed labels
UDP flood	DoS
SYN flood	
MAC flood	
Smurf	
Tamper	MITM
Replay	
ARP spoofing	
Benign	Benign

3.3. Detection Method: Dynamic Temporal Convolutional Network. As for the detection method, on the basis of the baseline TCN architecture [25], we conduct several architectural optimizations to build a basic TCN architecture that is good at dealing with temporal network packet sequences. Considering the basic TCN architecture has the disadvantage of computational waste when facing packets sequences with changeable temporal relationships, we adopt the adaptive computation time algorithm to optimize the architecture. In this way, a dynamic network that automatically adjusts the number of TCN blocks involved in the computation is finally built.

3.3.1. Optimized Temporal Convolutional Network. As a variation over the convolutional neural network, TCN is a descriptive term of a family of architectures aiming to solve temporal sequence prediction and classification issues. The prototype of TCN was proposed by Lea et al. between 2016 and 2017 [40, 41] and was further described and evaluated by Bai et al. in 2018 [25].

The commonly used DL methods for temporal sequence modelling are family of RNN methods, including vanilla RNN, LSTM, and GRU. However, as described in Section 2.1, these methods face some problems like being difficult to train. Thanks to the mechanism of convolutional neural network and its parallel computing process, TCN is easy to train and converges fast. Meanwhile, experiments carried out by [25] indicate that TCN performs better and is more effective across diverse sequence modelling tasks than RNN family architecture. So, we choose to use TCN as our basic method to build our DyTCN-IDS.

TCN is capable of processing temporal sequence with any length (once the model is constructed, this length is fixed). Formally, given an input temporal sequence $x_{1:T} = (x_1, x_2, \ldots, x_T)$, a TCN is a function $x^T \longrightarrow y^T$ producing the mapping:

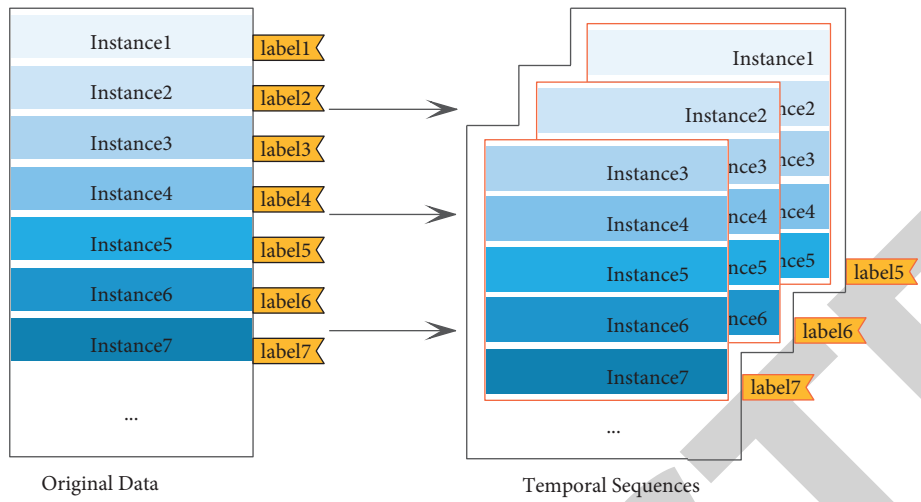

Figure 6: Construction process of the temporal sequence.

$$y_T = f(x_1, x_2, \ldots, x_T), \tag{2}$$

which indicates that the output y_T contains the information of x_T itself and information of temporal relationships between x_T and $(x_1, \ldots, x_{T-1})$. A simplified diagram of TCN is shown in Figure 7.

Four types of modern convolutional architectures, i.e., 1D convolutional network [42], causal convolution [43], dilated convolution [43], and residual connection [44], are integrated to form the baseline TCN.

(1) 1D convolutional network. It differs from the well-known CNN, 2D convolutional network, which uses a 2D convolution kernel (filter) moving by rows and columns to extract features from 3D data, and a 1D convolutional network utilizes a 1D kernel to extract features along the sequence direction, as shown in Figure 8. To keep the length of all hidden layers the same as the input layer, zero padding is adopted.

(2) Causal convolution. TCN adopts causal convolutions to maintain a strict causal time relationship, which means that the output y_T only contains the information convolved with elements no later than time T in the previous layer, as illustrated in Figure 9(a).

(3) Dilated convolution. There is a weakness of the simple causal convolution that its receptive field is linearly related to the depth of the network. This indicates that when facing a long temporal sequence, a very deep network with high computational cost is needed. To overcome this weakness, a dilated convolution illustrated in Figure 9(b) enables an exponentially large receptive field by applying a filter skipping input values with a certain step d (dilation rate). Formally, for a filter f and a sequence input $x_n \in (x_1, x_2, \ldots, x_T)$, the dilation operator F_d on input x_n is given by

$$F_d(n) = \sum_{i=0}^{k-1} f(i) \cdot x_{n-i \cdot d}, \tag{3}$$

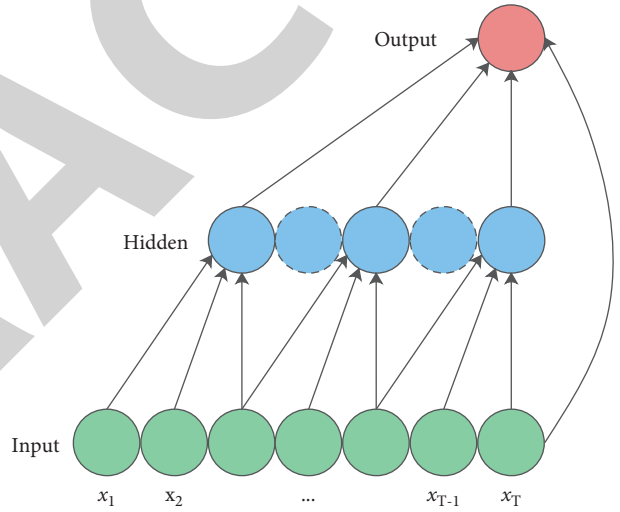

Figure 7: Simplified diagram of TCN.

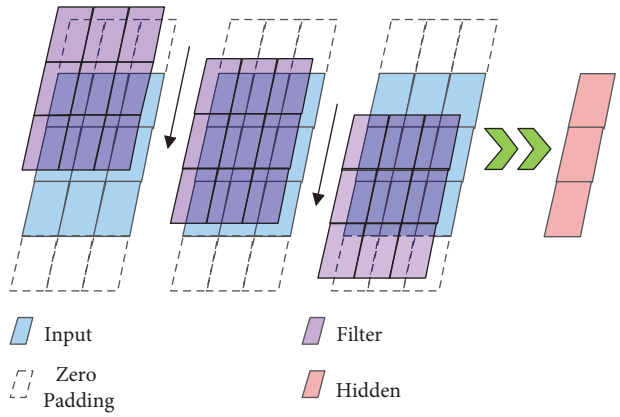

Figure 8: Convolution process of a 1D convolutional network.

where k stands for the filter size, d stands for the dilation rate, and $s - i \cdot d$ indicates the time direction of the front.

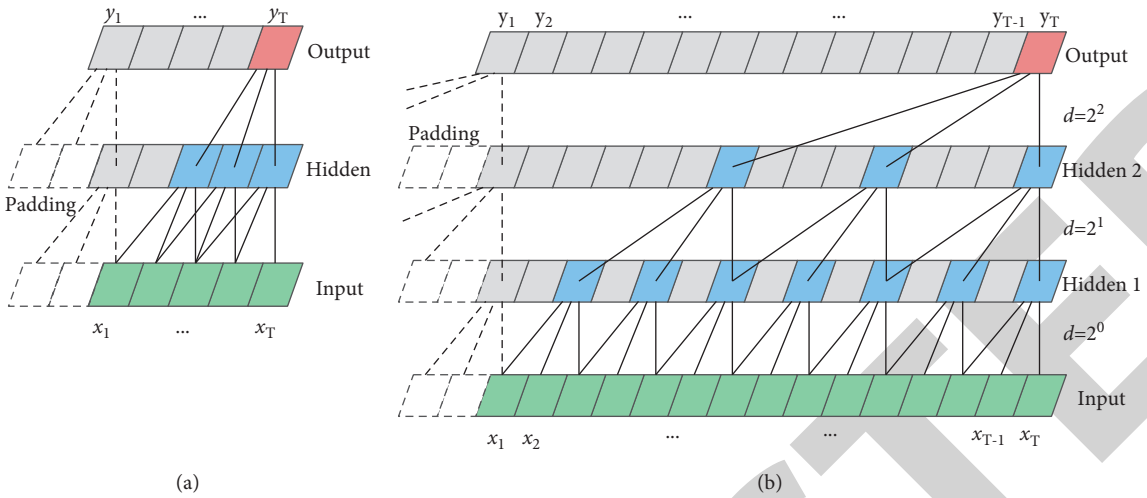

Figure 9: Comparison of a simple causal convolution with a dilated causal convolution. (a) A causal convolution with filter size $k = 3$. (b) A dilated causal convolution with filter size $k = 3$ dilation rate $d = 1, 2, 4$.

When there are L hidden layers, the receptive filed size RFS of a dilated convolution is calculated by

$$\mathrm{RFS} = (k-1) \cdot \sum_{i=1}^{L} d_L + 1, \tag{4}$$

where d_L is the dilation rate of the Nth hidden layer. Commonly, d_L is increased exponentially along with the depth of the network, as calculated by

$$d_L = 2^{L-1}. \tag{5}$$

This method of calculation ensures that, at the first hidden layer, where $d = 1$, all the information of inputs is extracted without losing, because filters do not skip here. Meanwhile, it also provides a large receptive field without too many hidden layers, because of the exponential relationship.

(4) Residual connection. As shown in formula (4), the receptive field of a TCN depends on the dilation rate d_L, the filter size k, and the number of hidden layers (network depth). In order to learn long temporal dependence, the depth of a TCN should increase. For example, for a sequence with a length of about 1000, a network with about ten hidden layers is needed. It is important to adopt appropriate means to maintain the stabilization of the network (avoiding network degeneracy and gradient vanishing), and residual connection is a good choice. As depicted in Figure 10, a residual connection uses a branch to achieve an identity mapping $f(x) = x$. The output of the network can be rewritten as

$$y = F_r(x) = g(F_d(x) + x), \tag{6}$$

where $F_r(x)$ is the residual function, $g(\cdot)$ is an activation function, $F_d(x)$ is the original dilated transformation, and y and x stand for the output and the input.

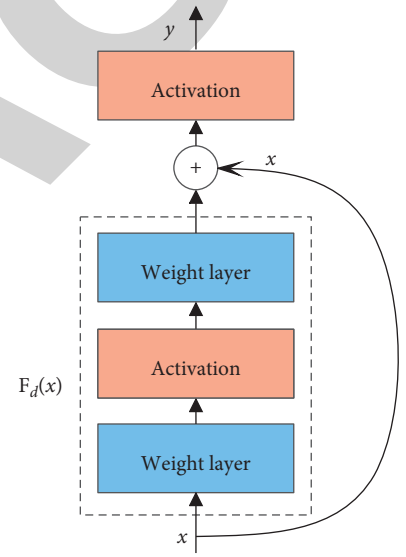

Figure 10: Residual connection using an identity mapping.

Through an elaborate combination of the above architectural elements, we designed the basic temporal convolutional network of our DyTCN-IDS, as illustrated in Figure 11. A stack of four TCN blocks makes up our basic TCN architecture with two dilated causal convolution layers in each TCN block (why four is chosen as the stack number will be explained in the experimental section). In order to ensure that the shortcut connection and the affine transformation have the same output size, a 1×1 convolution is always utilized in our method instead of the identity mapping. Same as the baseline TCN, weight normalization [45] is applied after each dilated causal convolution layer.

In addition, we make several architectural optimizations on the baseline TCN proposed by [18] to form our final basic TCN architecture:

(1) Unlike the baseline TCN, we do not use dropout after each dilated causal convolution layer. The

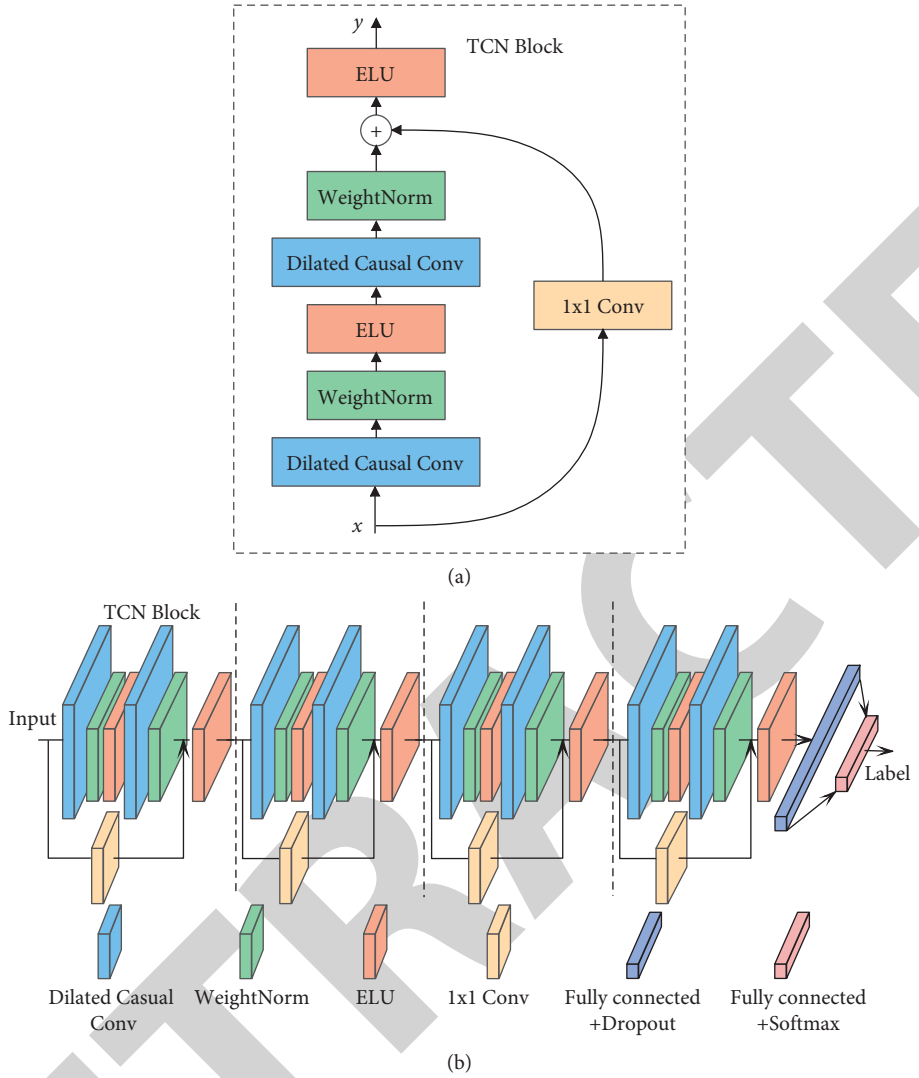

Figure 11: Basic TCN architecture of our DyTCN-IDS. (a) A TCN block of our basic TCN. (b) Four TCN blocks constitute the overall basic TCN architecture. Eight dilated causal convolution layers are used with filter dilation rate $d = 1, 2, 4, 8, 16, 32, 64, 128$.

reason for this is that a convolution layer does not have as many parameters as a fully connected layer, and therefore, using dropout after each convolution layer does not have much impact on the generalization ability of the model. In our basic TCN, dropout is adopted only between the last fully connected layer and the softmax function, which can achieve the same performance as the baseline TCN by a more compact architecture [46].

(2) Within a TCN block of our basic TCN, the activation function is added after the addition of the second convolution layer and the shortcut connection, which is inspired by the well-verified Resnet [37]. The baseline TCN adds an activation function before the addition, which does not express a clear meaning.

(3) The activation function used in the baseline TCN is ReLU [47], which faces the dying ReLU problem. We adopt exponential linear unit (ELU) [41] to replace ReLU. ELU is an approximate zero-centered nonlinear activation function and is more robust to input changes or noise.

To sum up, firstly, we use dropout between the last fully connected layer and the softmax function in a TCN block to build a compact and effective architecture. Secondly, inspired by the well-known Resnet, we set the activation function after the addition of the second convolution layer and the shortcut connection. Thirdly, we adopt ELU to replace the original ReLU activation function of baseline TCN to overcome the dying ReLU problem. Experimental results in Section 4.3.3 indicate that after our architectural optimizations are adopted, our TCN acquires a better detection performance than the baseline TCN and avoids the overfitting problem.

Our method constructs a series of consecutive network packets into temporal sequences like described in Section 3.2.5. The temporal sequences are the input to DyTCN, and temporal relationships between network packets are learned

by performing a sequential one-dimensional dilated casual convolution between the samples within the sequence. The longer the sequences are, the more convolution layers are needed, and the larger the kernel size is.

3.3.2. Dynamic Optimization of TCN. Compared to baseline recurrent architectures, TCN appears not only more accurate, but also simpler and easier to train. However, TCN still has one obvious drawback when used in intrusion detection issues. That is, once the depth of the network is selected, no matter how long the temporal relationship in the input sequence is related to x_T, the network traverses the entire receptive field. For network packet sequences used as the input of our system, the length of the temporal relationships is changeable, which means that we need to create a long packet sequence to cover the longest temporal sequence. This kind of sequence construction method ensures that all temporal relationships are covered but also causes a computational waste when faced with a short temporal relationship, as shown in Figure 12.

Assuming the filter size k is selected, the receptive field RFS is determined by the depth of the network. The deeper the network, the bigger the perceptive field. Imagine that if the network can skip the rest convolution layers when enough temporal correlative information has been extracted, the aforementioned computational waste will be greatly reduced. To reach this goal, dynamic neural network technologies [48] could be a good choice. Adaptive computation time (ACT) [49] is a layer skipping algorithm that allows RNN to learn how many computational steps can be taken between receiving an input and emitting an output. Inspired by this work, we adopt the ACT algorithm to achieve a dynamic neural network optimization to our basic TCN architecture, so as to complete our whole DyTCN method.

We add a branch to the output of each TCN block to predict a halting score h which is in the range $[0, 1]$. Halting scores are calculated sequentially as shown in Figure 13. Once the cumulative sum of h reaches one, all remaining TCN blocks will be skipped. To ensure that all the halting scores sum to one, the h of the last branch before the skipping is replaced by a remainder, which equals to one minus the former cumulative sum. The output of the TCN network is then redefined in the form of a weighted sum of the outputs of former TCN blocks, where the weight of each TCN block is given by a halting probability value. We use a ponder cost that is the number of calculated TCN blocks to make the computation more likely to stop earlier, that is, because minimizing the ponder cost increases the halting scores of former TCN blocks. The final loss function is the sum of the original loss function and summed ponder costs that are weighted by a constant τ. The entire formal calculation process is given below.

Considering our basic TCN architecture of M TCN blocks (experimental results show that $M = 4$ achieves the best results),

$$\begin{aligned} &\text{input} = x^0, \\ &x^m = g\left(F_d\left(x^{m-1}\right) + x^{m-1}\right), \quad m = 1, \ldots, M, \\ &\text{output} = x^M. \end{aligned} \tag{7}$$

The halting score $h^m \in [0, 1]$ for each TCN block is calculated by a sigmoidal function $H(\cdot)$ and the halting score of the last block h^M is set to 1 to enforce stopping after this block, as shown in the following:

$$\begin{aligned} &\sigma(x) = \frac{1}{1 + \exp(-x)}, \\ &h^m = H\left(x^m\right) = \sigma\left(W^m \cdot pool\left(x^m\right) + b^m\right), \quad m = 1, \ldots, M-1, \\ &h^M = 1, \end{aligned} \tag{8}$$

where $\sigma(\cdot)$ is a logistic function, and $\text{pool}(\cdot)$ is a global average pooling [50].

The number of blocks to calculate is N and it is given by

$$N = \min\left\{n: \sum_{m=1}^{n} h^m \geq 1 - \varepsilon\right\}, \tag{9}$$

where ε is a very small constant ensuring that N can be equal to 1, which means that the computation can be halted after a single block.

The remainder R is defined by

$$R = 1 - \sum_{m=1}^{N-1} h^m. \tag{10}$$

Furthermore, the halting probability p is calculated by

$$p^m = \begin{cases} h^m, & m < N, \\ R, & m = N, \\ 0, & m > N. \end{cases} \tag{11}$$

The new output of the dynamic TCN architecture is updated to the sum of the outputs of TCN blocks weighted by the halting probabilities, as given by

$$\text{output}' = \sum_{m=1}^{N} p^m x^m. \tag{12}$$

Our goal is to encourage the computation to stop earlier, i.e., to minimize the number N. However, N is a non-differentiable discrete variable that is unable to be optimized with gradient descent. The ponder cost, an almost differentiable upper bound of N, is introduced here to help achieve this goal. It is calculated by

$$\rho = N + R. \tag{13}$$

The pseudocode description of the calculation process of ACT in our method is shown in Algorithm 1.

Using the output ponder cost of Algorithm 1, we can improve the backpropagation process of our method. When differentiating the ponder cost, the gradient of N can be

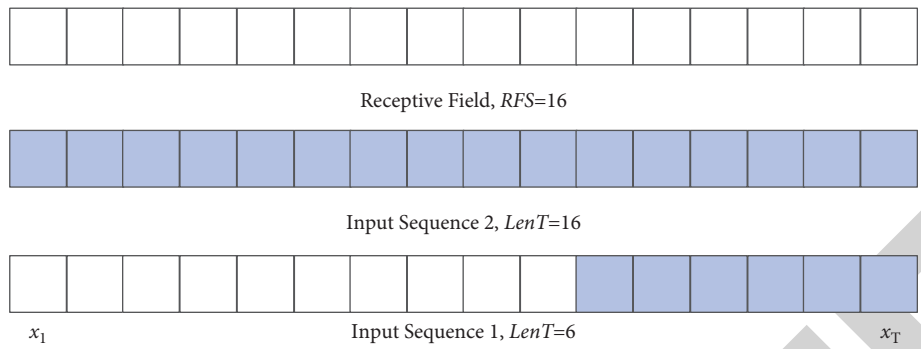

Figure 12: Comparison of different length Len T of temporal sequences within an input packet sequence to the receptive field. Input sequence 2 has the longest temporal relationship 16, so RFS is set to 16 to cover it. Facing the input sequence 1 with Len $T = 6$, information of $(x_1, x_2, \ldots, x_{10})$ is not useful and the computational cost on these ten packets is wasted.

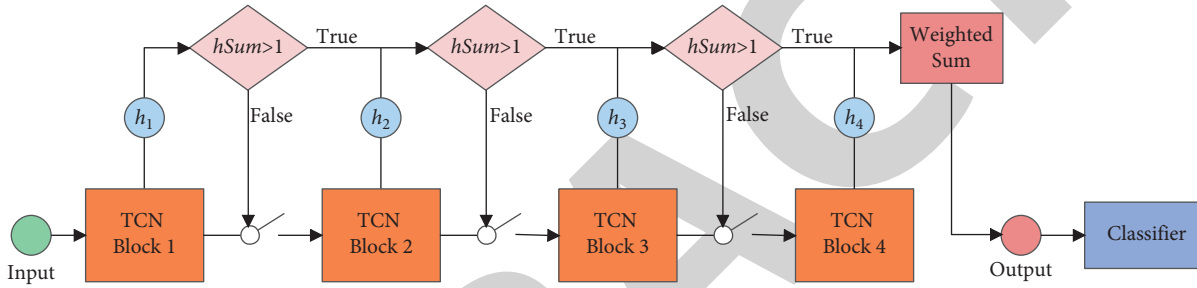

Figure 13: Dynamic layer skipping of DyTCN using APC algorithm. After each TCN block, a halting score h is calculated, and a judgement of whether the accumulative sum h Sum of h exceeds one is conducted. Once the judgement is true, the rest of TCN blocks will be skipped. The halting score of the last TCN block is fixed to one to enforce stopping here. The final output is a weighted sum, as seen in equation (12).

Input: network packets sequences **input**
Input: number of TCN blocks $M = 4$
Output: 2-D tensor **output**
Output: ponder cost ρ
(1) **input** = x
(2) **output** = 0
(3) sum = $0, \rho = 0, R = 1$
(4) **for** $m = 1, \ldots, M$ **do**
(5) $x = F_r^m(x)$
(6) **if** $m < M$ **then** $h = H^m(x)$
(7) **else** $h = 1$
(8) **end if**
(9) sum+ = h
(10) ρ+ = 1
(11) **if** sum $< 1 - \varepsilon$ **then**
(12) $R-$ = h
(13) **output** += hx
(14) **else**
(15) ρ+ = R
(16) **output** += Rx
(17) **break**
(18) **end if**
(19) **end for**
(20) **return output**, ρ

Algorithm 1: The dynamic optimization of our basic TCN architecture using ACT.

ignored according to [49] and the partial derivative of ρ with respect to a halting score h^m is given by

$$\frac{\partial \rho}{\partial h^m} = \begin{cases} -1, & m < N, \\ 0, & m \geq N. \end{cases} \tag{14}$$

Minimizing ρ will increase $h^1, \ldots, h^{m-1}$, which will enable the computation halts earlier. The whole DyTCN is trained with the final loss function:

$$\ell' = \ell + \tau\rho, \tag{15}$$

where $\tau \geq 0$ is a time penalty parameter (acts as a hyperparameter in our method), and it weights the relative computational cost versus error. The final loss function $\ell\prime$ is differentiable, so that it can be optimized by backpropagation.

To sum up, as described above, because the length of the temporal relationships of temporal attacks against TCE are changeable, and the length of input sequence of our basic TCN is fixed, the computational resources will be wasted when the former length is smaller than the latter. APC algorithm-based dynamic neural network technology is capable of skipping network layers dynamically when enough information is captured from a changeable input sequence. To overcome this problem, we use the APC algorithm-based dynamic neural network technology to optimize our basic TCN architecture and finally build a dynamic network for intrusion detection, which is the aforementioned DyTCN. The main idea of DyTCN is to take advantage of the merits of TCN in processing temporal sequences, while combining with a dynamic technology to decrease the computational cost. Next, several experiments will be conducted to evaluate the performance of our DyTCN-IDS.

4. Experiments and Results

In this section, evaluation metrics suitable for DL-based intrusion detection models are first introduced. Detailed experimental settings are given next. Experiments are conducted to determine the best M and τ. Then, the capability of our dynamic optimization method to reduce computational cost is verified. Finally, the detection performance of our DyTCN-IDS is evaluated through comprehensive experiments.

4.1. Evaluation Metrics. There are five most widely used evaluation metrics: detection accuracy Acc, recall R, precision P, F-score F, and false alarm rate FAR. Among them, F is a summary expression for P and R. Therefore, we choose Acc, FAR, and F, as our evaluation metrics.

All the metrics are calculated on the basis of the confusion matrix, which is a 2D matrix providing classification results about the actual and predicted class. There are four attributes used in the confusion matrix:

(1) True positive (TP): network packets correctly predicted as attacks

(2) False positive (FP): network packets wrongly predicted as attacks

(3) False negative (FN): network packets wrongly predicted as benign packets.

(4) True negative (TN): network packets correctly predicted as benign packets

The meaning of the five metrics is as follows:

(1) Accuracy Acc: it is the ratio of correctly predicted instances to the total instances, that is,

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}. \tag{16}$$

(2) Recall R: it is the ratio of correctly predicted attacks to all the ground-truth attack instances, that is,

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{17}$$

(3) Precision P: it is the ratio of correctly predicted attacks to the instances classified as attacks, that is,

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{18}$$

(4) False alarm rate FAR: it is the ratio of wrongly predicted attacks to all the benign instances, that is,

$$\text{FAR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{19}$$

(5) F-score F: it is a weighted harmonic mean of precision and recall, that is,

$$F = \frac{\left(1 + \beta^2\right) \cdot P \cdot R}{\left(\beta^2 \cdot P\right) + R}, \tag{20}$$

where β measures the relative importance of precision and recall. For intrusion detection issues, recall that reflecting the degree of missing report of attacks is more important than precision. Therefore, we set $\beta = 1.2$ to make the F-score more likely to measure the recall.

The above metrics are used to evaluate binary classification; however, our experiments are carried out on multiclass classification issues. To solve this, the multiclass classification issue can be treated as a combination of several binary classification problems. The macro average is adopted to evaluate the multiclass classification performance globally due to its advantage of being capable of paying more

attention to the classes with small sample capacity. The macro average is calculated by

$$\begin{cases} \text{macro}P = \dfrac{1}{n}\sum_{i=1}^{n} P_i, \\ \text{macro}R = \dfrac{1}{n}\sum_{i=1}^{n} R_i, \\ \text{macroFAR} = \dfrac{1}{n}\sum_{i=1}^{n} \text{FAR}_i, \\ \text{macro}F = \dfrac{\left(1+\beta^2\right)\cdot \text{macro}P \cdot \text{macro}R}{\left(\beta^2 \cdot \text{macro}P\right) + \text{macro}R}, \end{cases} \tag{21}$$

where $n = 3$ is the number of classes in this paper.

Besides Acc, FAR, and F, we adopt floating point operations (FLOPs), which are a resource-aware metric to evaluate the model complexity and thus evaluate the ability of our dynamic optimization to reduce computational cost.

4.2. Experimental Settings. Experiments were carried out using the following hardware and software platforms: AMD Ryzen 5 1600X Six-Core processor @3.60 GHz, 32 GB RAM, NVIDIA GeForce GTX 1060 6 GB GPU, Windows 10 operating system, Keras 2.5.0, and CUDA 10.2.95.

Our DyTCN is specifically designed for detecting temporal attacks against TCE. However, other commonly used public network attack datasets are not captured in the TCE scenario, and the TCE scenario is quite a special industrial network scenario, which means that these datasets have different characteristics with TCE network data and do not reflect the real TCE situation. So, we choose not to use these public datasets. Because our semiphysical TCE testbed is built based on a certain type of TCE train, and all the benign and attack packets are implemented according to actual situations of TCE, we believe that our TCE-IDS dataset captured from this testbed can well simulate the real TCE situation. So, we finally choose to use TCE-IDS dataset to train and test our method.

TCE-IDS dataset has a total number of 944860 instances, including 363818 DoS attack instances, 155063 MITM attack instances, and 425979 benign instances. In order not to break the original temporal relationships, we separate our dataset into training, validation, and test subset after the feature engineering part. Using the stratified 3-way holdout method that ensures the original class proportion, the dataset was spilt out into three subsets: 60% for training, 20% for validation, and 20% for test.

Hyperparameters used in our system including two categories. The first category is normal hyperparameters commonly used in deep learning method, including batch size, training epoch, learning rate, filter size, and filter number. The second category is special hyperparameters specific for our method, including TCN blocks number M, time penalty parameter τ, and sequence length *seqLen*.

Normal hyperparameters are set according to the rule of thumb. Special hyperparameters are set according to tuning experiments described in Section 4.3.1. Table 3 presents the hyperparameter configuration. Weights and biases in our system are initialized randomly on the basis of Gaussian distribution. Adam algorithm is adopted as the optimizer.

4.3. Results and Analysis

4.3.1. Parameter Tuning Experiments for Special Hyperparameters. As described in Section 4.2, there are three special hyperparameters used in our method. The values of them are set according to tuning experiments, as described in the following:

(1) Experiments for Finding the Best. M (The Number of TCN Blocks to be Stacked). In Section 3, we described that our system uses four stacked TCN blocks to form our basic TCN architecture. How this $M = 4$ is elected through experiments is described here.

We set the initial test range $M = [2, 3, 4, 5, 6]$ to build five basic TCN models. According to equation (4), networks with such TCN blocks have the receptive field size of $[31, 127, 511, 2047, 8095]$. Accordingly, we fed different input packet sequences with seq Len = $[30, 120, 500, 2000, 8000]$ from TCE-IDS dataset into these five models. Experimental results are shown in Figure 14.

As shown in Figure 14, although the computational cost reflected by FLOPs is low when M is two and three, it does not reach the optimal *macroF*. When M is larger than four, the computational cost increases rapidly without facilitation on *macroF*. $M = 4$ reaches the highest detection performance and bears a relatively low computational cost, which means that this value acquires the best trade-off between detection performance and computational cost. So, we choose to build our basic TCN architecture with four TCN blocks.

(2) Experiments for Finding the Best. τ (The Time Penalty Parameter). The time penalty parameter τ is an important hyperparameter of our dynamic optimization. It values the trade-off between the original basic TCN architecture and the dynamic skipping unit. Furthermore, it measures the propensity of the dynamic part to skip layers. It is first used in intrusion detection issues with a temporal convolutional network by our work, so there is not much experience to follow. Experiments of our DyTCN were conducted to search for a good τ that reconciles both detection performance and computational cost, as illustrated in Figure 15.

As shown in Figure 15, computational cost shows an overall downward trend with the increase of τ. This is because a larger τ enables the dynamic part more inclined to skip earlier. The detection performance *macroF* also reveals a downward trend but to a slighter extent. The reason of this could be that the earlier the network skip layers, the fewer the high-dimensional features extracted. As a trader-off between *macroF* and FLOPs, we choose $\tau = 0.015$ for our

Table 3: Hyperparameter configuration.

Hyperparameter	Value	Setting principle
Batch size	64	
Training epoch	20	
Learning rate	0.002	Rule of thumb
Filter size k	3	
Filter number	64	
TCN blocks number M	**4**	
Time penalty τ	**0.015**	**Tuning experiments**
Sequence length seq Len	**480**	

Bold values are special hyperparameters used in our method and are described in Section 4.3.1.

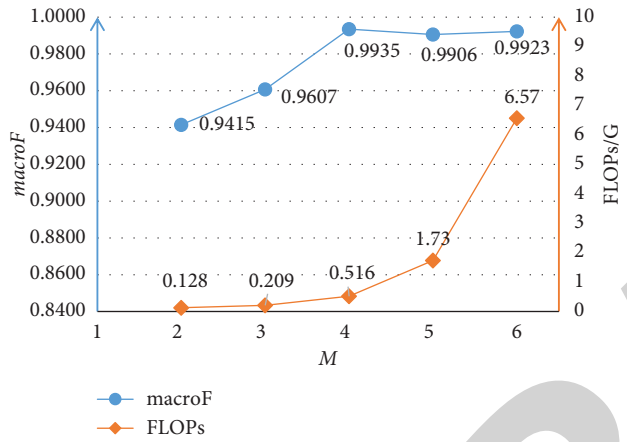

Figure 14: Comparison of different models with $M = [2, 3, 4, 5, 6]$ TCN blocks: mean value from 10 repetitions.

DyTCN model, where *macroF* is highest (0.9939), and FLOPs is acceptable (0.415 G).

(3) Experiments for Finding the Best seqLen (Length of the Constructed Temporal Sequence). As we have selected to use four TCN blocks as described above, the maximum input sequence length is 511, which means that the network packets constructed in one input temporal sequence should be less than 511. We have tested that when M is two, and *seqLen* is 120, the *macroF* is lower, which precludes choosing a length of 120 or less. From 120 to 500, we tested different length at an interval of 20, and the experimental results are as Figure 16.

It can be seen that, in Figure 16, as the sequence length increases, the *macroF* shows an increasing trend. This means that, in the range of 120 to 500, the larger the *seqLen* is, the higher the detection performance is. According to theory analysis, longer sequence can contain more temporal relationship information and contributes more in the detection process. So, the experimental results are consistent with the theoretical analysis. Of course, longer is not always better, and this depends on how much the temporal attacks disrupt the normal temporal relationship of benign packets. For our TCE-IDS dataset, through experiments carried out in 4.3.1(1), we can consider that sequences longer than 500 do not carry more temporal relationship information, and this is also why we choose to tune *seqLen* in range of smaller than 500.

The best *macroF* performance is acquired at *seqLen* of 460 or 480 or 500. We choose to use 500 for two reasons. The first reason is that 500 is easier to handle as a more standard integer. The second reason is that, benefiting from our dynamic neural network design, the length of the input sequence can be dynamically downward and compatible. So, we choose the maximum value 500 in the optional range (460, 480, 500).

4.3.2. Model Training Experiments. Our DyTCN intrusion detection model was trained on our TCE-IDS dataset and was evaluated by the mentioned evaluation metrics. Figures 17(a) and 17(b), respectively, illustrate the detection accuracy curve and loss curve on training set and validation set. Table 4 shows the test results of DyTCN on test set.

It can be seen from Figure 17(a) that the training loss and validation loss decrease rapidly in the first few epochs. This means that the model is learning fast, and usable information is captured. Afterward, losses are falling at a slower rate, and this means that the model is learning the harder part of information in training data. In the later stage of training, the loss tends to stabilize, and its value is small. From Figure 17(b), we can also see a consistent trend that the training accuracy and validation accuracy rise rapidly and then stabilize at about 99.4%. These results show that the DyTCN has fast convergence speed, small training error, and high accuracy. Another situation that can be seen is that the validation loss is almost equal or slightly higher than the training loss, and the validation accuracy is almost equal or slightly less than the training accuracy. This means that DyTCN is able to overcome overfitting very well.

Test results in Table 4 indicate that DyTCN has both low false alarm rate and high F-score ($\beta = 1.2$) on DoS attacks and MITM attacks. The overall macro false alarm rate is 0.09% and means that DyTCN has a good performance of not misclassifying benign packets as attacks. The overall macro F-score and accuracy are 99.39% and 99.40%, which means that DyTCN has a promising overall detection performance on TCE-IDS dataset.

4.3.3. Comparison Experiments

(1) Comparison Experiments with Baseline TCN and Basic TCN. Experiments were carried out on TCE-IDS dataset to compare the detection performance between baseline TCN, our basic TCN, and our DyTCN. Table 5 shows their evaluation results. Experiments for evaluating the ability of our dynamic optimization to reduce computational cost are also conducted, and Figure 18 shows the computational cost between our basic TCN and DyTCN.

It can be seen from Table 5 that our basic TCN and DyTCN perform better than baseline TCN on all the evaluation metric, which proves that our architectural optimizations and dynamic neural network optimization acquire promising effect. According to theory analysis, the detection ability of our DyTCN should be weaker than that of the basic TCN. This is because we think that the dynamic

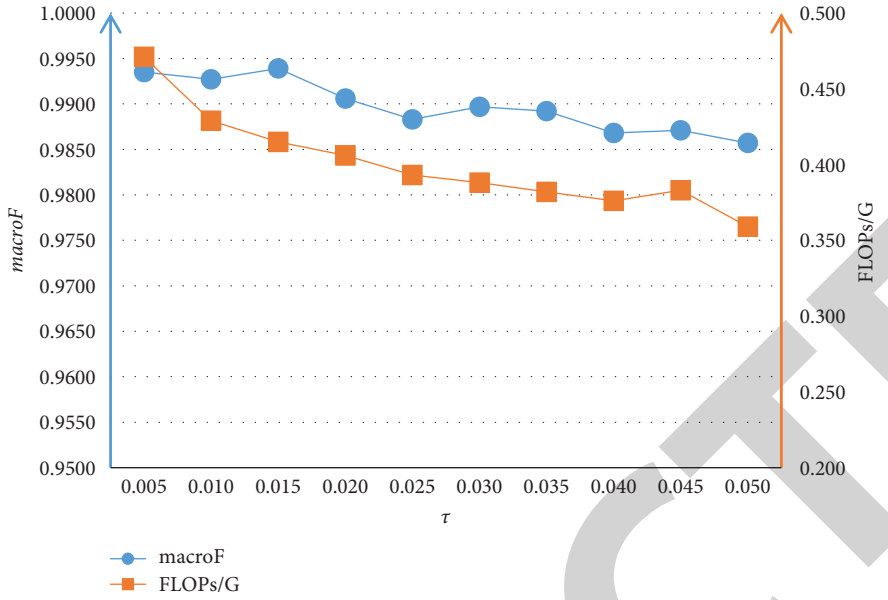

Figure 15: Comparison of different τ in our DyTCN: mean value from 10 repetitions.

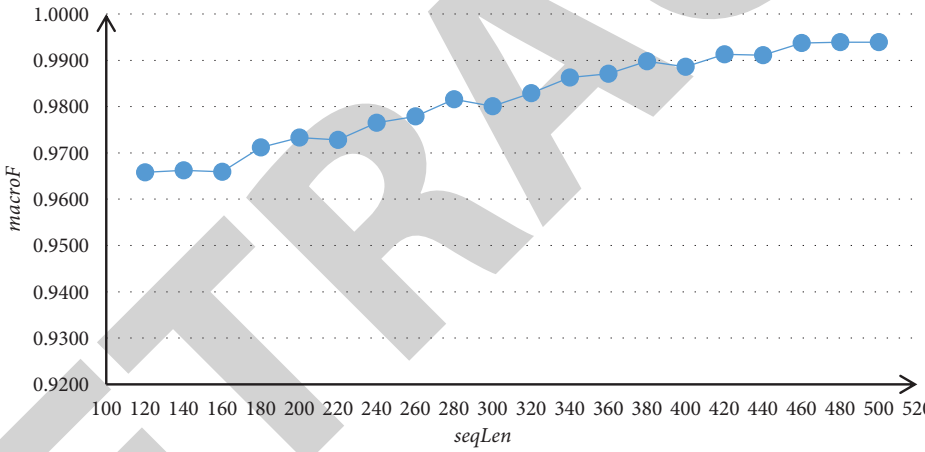

Figure 16: Comparison of different *seqLen* in our DyTCN: mean value from 10 repetitions.

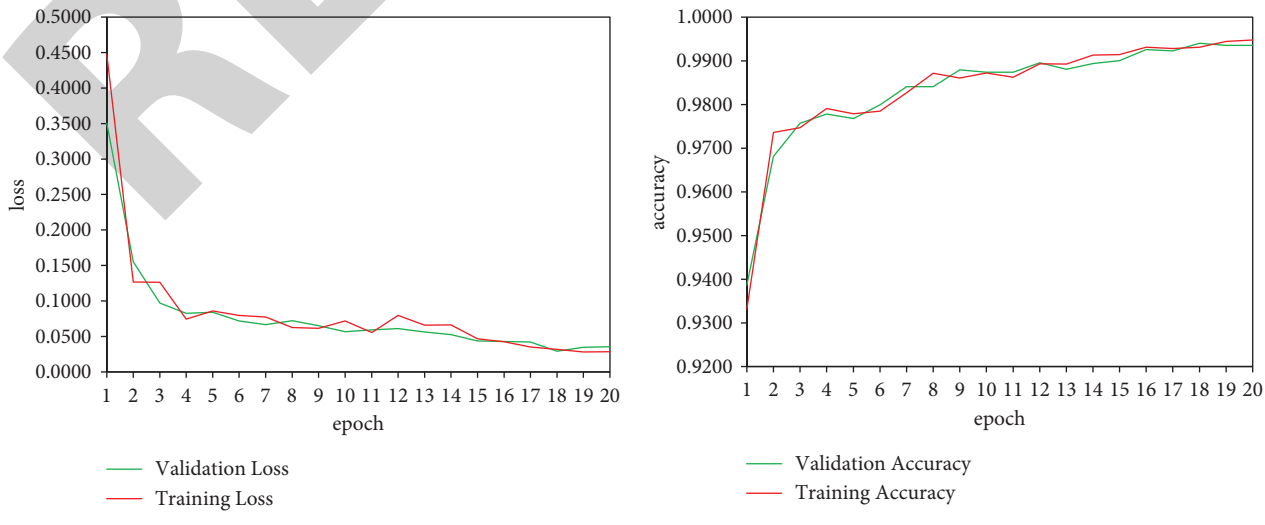

Figure 17: Training graphs of DyTCN on TCE-IDS dataset: mean value from 10 repetitions: (a) loss curve; (b) accuracy curve.

Table 4: Test results of DyTCN on TCE-IDS dataset: mean value from 10 repetitions.

	DoS	MITM	Overall
FAR	0.0007	0.0009	—
F	0.9941	0.9943	—
macroFAR	—	—	0.0009
macroF	—	—	0.9939
Acc	—	—	0.9940

Table 5: Test results of baseline TCN, our basic TCN, and DyTCN on TCE-IDS dataset: mean value from 10 repetitions.

Models	DoS		MITM		Overall		
	FAR	F	FAR	F	macroFAR	macroF	Acc
Baseline TCN	0.0015	0.9843	0.0012	0.9841	0.0014	0.9843	0.9847
Our basic TCN	0.0008	0.9938	**0.0007**	0.9933	**0.0008**	0.9935	0.9937
Our DyTCN	**0.0007**	**0.9941**	0.0009	**0.9943**	0.0009	**0.9939**	**0.9940**

Bold values are the best values in experimental results.

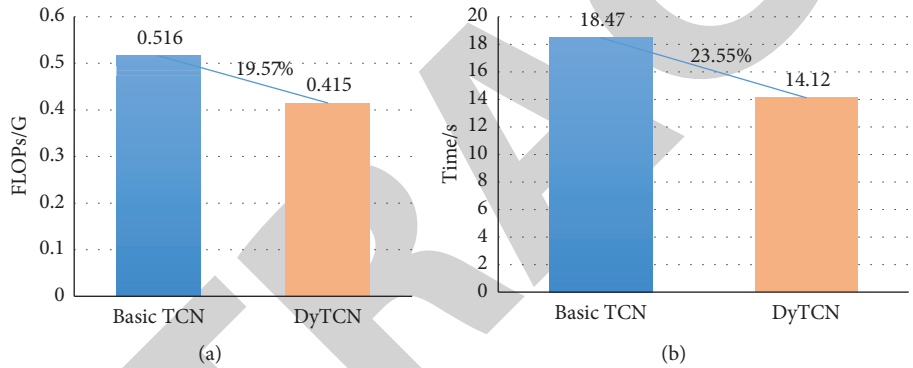

Figure 18: Comparison of FLOPs and inference time (10000 instances) of basic TCN and DyTCN: mean value from 10 repetitions: (a) FLOPs comparison; (b) inference time comparison.

optimization costs less computational resources, so that the feature extraction ability of the model will degrade a little. However, an interesting finding is that our DyTCN performs even slightly better than our basic TCN while consuming fewer computational resources. This may be due to the dynamic adaptability of the model to fully extract the features of the input sequences while avoiding overfitting. And this further proves that our dynamic design has the capability of maintaining promising detection capabilities while reducing computational cost.

The purpose of our dynamic optimization is to automatically adjust the depth of the network according to the different length of the time relationship implied by the input sequence so as to reduce the computational cost. Experiments were conducted using our basic TCN and the whole dynamic TCN separately. We tested inference time (time spent on test) between them using 100000 instances. We also recorded the average FLOPs required for each input sequence in the test set. The results presented in Figure 18 show that when facing input sequences with different inner temporal relationship length, DyTCN costs less computational resources and inference time than basic TCN. To be specific, 19.57% of computational resources and 23.55% of time are saved by our dynamic optimization on our basic TCN architecture.

(2) Comparison Experiments with Canonical DL-Based Intrusion Detection Methods. In order to verify the intrusion detection superiority of DyTCN, comparison experiments with canonical DL methods with several canonical DL-based intrusion detection methods were conducted on TCE-IDS dataset. The selected comparison methods include two classes. The first class is methods that do not concern temporal relationships, containing MLP (Multi-Layer Perceptron) and VGG11. The VGG11 model uses the hyperparameters of the original text. The VGG11 model uses the hyperparameters of the original text [51]. The second class is methods that concern temporal relationships, containing LSTM and GRU. They use one layer with 256 hidden units. Test results of them are shown in Table 6. Overall results are illustrated in Figure 19(a) for a better visual comparison.

From Table 6 we notice that MLP and VGG11 have poor detection performance, and this is because they cannot capture temporal features. From another point of view, MLP

Table 6: Test results of comparison experiments with canonical DL-based intrusion detection methods: mean value from 10 repetitions.

Models	DoS		MITM		Overall		
	FAR	F	FAR	F	macroFAR	macroF	Acc
MLP	0.0114	0.8620	0.0153	0.7870	0.0135	0.8355	0.8427
VGG11	0.0083	0.9213	0.0137	0.8111	0.0111	0.8846	0.9037
LSTM	0.0057	0.9593	0.0079	0.9556	0.0069	0.9571	0.9581
GRU	0.0051	0.9619	0.0075	0.9596	0.0063	0.9607	0.9612
DyTCN	**0.0007**	**0.9941**	**0.0009**	**0.9943**	**0.0009**	**0.9939**	**0.9940**

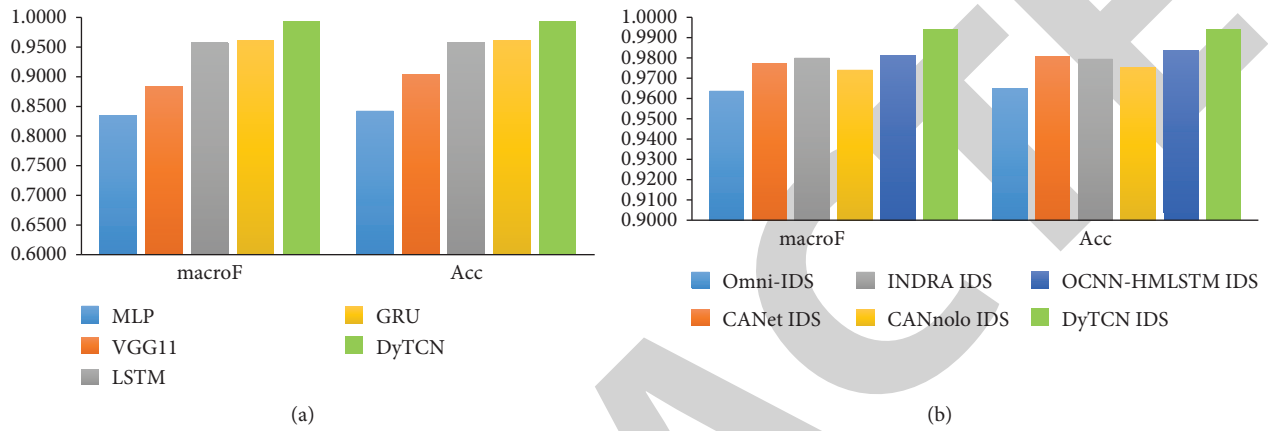

Figure 19: Overall comparison results of *macroF* and *Acc*: mean value from 10 repetitions: (a)comparison with canonical DL methods; (b) comparison with latest IDSs.

and VGG11 still have some detection capability, and this means that temporal attacks still have some spatial features rather than only temporal relationships. Because MITM attacks contain more temporal features and less spatial features, their performance on MITM is worse than that in DoS. LSTM and GRU models that are often used in sequence modelling tasks achieve a mediocre performance. This proves that these two methods can be used in intrusion detection tasks of TCE but cannot reach a promising performance. From Table 6 and Figure 19(a) we can see that our DyTCN acquires the best performance, which proves that DyTCN is suitable for detecting temporal attacks against train communication Ethernet than those canonical DL methods.

(3) Comparison Experiments with Latest Temporal Relationship Concerned IDS. In order to verify that DyTCN can better detect temporal attacks against TCE, comparison experiments with latest temporal relationship concerned IDSs (in range of 2020 to 2021) were conducted on TCE-IDS dataset. The selected temporal relationship concerned IDS contains Omni IDS (FNN and LSTM based) [23], CANet IDS (LSTM and Autoencoder based) [52], INDRA IDS (GRU and Autoencoder based) [53], CANnolo IDS (LSTM and Autoencoder based) [54], and OCNN–HMLSTM IDS (CNN and LSTM based) [55].

Because some of these IDSs are binary anomaly detection-based IDS, the comparison experiments use overall metrics (viz. *macroFAR*, *macroF*, and *Acc*) to evaluate their performance. The test results are listed in Table 7. For a better visualization, we illustrate the overall results of *macroF* and *Acc* in Figure 19(b) (these two metrics can best reflect the comprehensive performance of an intrusion detection model).

It can be seen from Table 7 that latest IDSs have a not bad macro F-score ($\beta = 1.2$) and accuracy, which means that these IDSs have capabilities to deal with temporal attacks against TCE to some extent. However, compared with DyTCN, these IDSs cannot acquire promising detection performances. The first reason for this is that they are mainly recurrent neural network-based methods, and from the comparison experiments in 4.3.3(2), we can know that this class of methods performs worse than DyTCN. The second reason is that they are designed for other scenarios rather than train communication Ethernet scenarios. So, it is predictable that they do not perform well on our TCE-IDS dataset. Of course, due to their optimization design, they perform better than canonical RNN methods like LSTM and GRU. As can be seen in Table 7 and Figure 19(b), our DyTCN outperforms all the other models on TCE-IDS dataset. The *macroFAR* of DyTCN is 0.09%, and this means that DyTCN has a good capability to correctly recognize benign packets. The *macroF* and *Acc* of DyTCN are 99.39% and 99.40%, which means that DyTCN is able to detect DoS and MITM attacks against TCE very well.

To sum up, from the training experiments described above, we can see that the proposed DyTCN-IDS converges fast in training and can avoid overfitting. From all the comparison experiments, we can see that DyTCN has a superior intrusion detection performance than several

Table 7: Test results of comparison experiments with latest temporal relationship concerned IDS: mean value from 10 repetitions.

Models	macroFAR	macroF	Acc
Omni-IDS [23]	0.0057	0.9638	0.9653
CANet IDS [52]	0.0042	0.9772	0.9810
INDRA IDS [53]	0.0035	0.9801	0.9798
CANnolo IDS [54]	0.0049	0.9741	0.9755
OCNN–HMLSTM IDS [55]	0.0026	0.9785	0.9798
DyTCN-IDS	**0.0009**	**0.9939**	**0.9940**

Bold values are the best values in experimental results.

canonical DL-based IDSs and several latest IDSs. While acquiring promising detection performance on TCE-IDS dataset, the DyTCN can also cost less computational resources that canonical TCN due to our dynamic neural network optimization. In a word, all the results indicate that DyTCN is not only powerful in detecting temporal attacks against TCE, but is also computationally efficient.

4.4. Limitations. There are three limitations of our DyTCN-IDS, as described in the following:

(1) The proposed DyTCN-IDS is specifically designed for detecting temporal attacks in TCE, and the limitation is that its performance on other non-temporal attacks in TCE is not verified.

(2) The proposed DyTCN-IDS is tested in our TCE-IDS dataset, and the limitation is that whether it can achieve promising performance on other dataset in other scenarios is not verified.

(3) Our TCE-IDS dataset is captured in a self-built semiphysical TCE testbed. Compared with real data in real trains, the authenticity of data and the diversity of attack classes still have some limitations.

Although these limitations limit the application scope of our DyTCN-IDS, they also point out the direction for our further research work.

5. Conclusions

In this paper, a novel dynamic temporal convolutional network-based intrusion detection system is proposed to detect the temporal correlated DoS and MITM attacks against TCE. In order to establish a reasonable data environment for validation, a TCE testbed is built to generate a TCE attack dataset including several kinds of specific DoS attacks and MITM attacks (the dataset can generally mimic a real train scenario), and a systematic feature engineering is designed elaborately. Specifically, the detection method in our system consists of two main parts. In the first part, a basic TCN model was designed to capture temporal relationships between network packets. In the second part, a dynamic optimization of our basic TCN architecture using the APC algorithm was proposed to reduce the computational cost of the basic TCN model. The experimental results indicate that the proposed DyTCN-IDS is not only computationally efficient, but also yields a superior result in terms of macro false alarm rate (0.09%), macro F-score (99.39%), and accuracy (99.40%).

Network attack technologies are developing rapidly, so new kind temporal attacks are constantly emerging. The excellent results obtained in this paper encourage us to validate the effectiveness of the proposed system to new temporal attacks against TCE in the future. Our system is evaluated on the dataset obtained from our TCE testbed. In the future, we will further improve our system through field trials in a real train.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by Science and Technology Research and Development Key Program of China State Railway Group Co., Ltd. under Grant N2020J007.

References

[1] Electronic Railway Equipment-Train Communication Network—Part 1, "General architecture," *Document IEC*, vol. 61375, 2012.

[2] J. Moreno Garcia-Loygorri, J. Goikoetxea, E. Echeverría, A. Arriola, and I. Val, "The wireless train communication network: roll2Rail vision," *IEEE Vehicular Technology Magazine*, vol. 13, no. 3, pp. 135–143, 2018.

[3] J. Yin, S. Su, J. Xun, T. Tang, and R. Liu, "Data-driven approaches for modeling train control models: comparison and case studies," *ISA Transactions*, vol. 98, pp. 349–363, 2020.

[4] "UNIFE vision paper on digitalisation," 2021, https://www.unife.org/publication-press/publications/164-unife-vision-paper-on-digitalisation.html.

[5] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: a survey and an objective comparison," *Journal of Network and Computer Applications*, vol. 169, Article ID 102767, 2020.

[6] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijaylakshmi, L. Yogesh, and K. Arputharaj, "Intelligent feature selection and classification techniques for intrusion detection in networks: a survey," *EURASIP Journal on*

Wireless Communications and Networking, vol. 271, no. 1, pp. 1–16, 2013.

[7] Z. A. Khan and P. Herrmann, "Recent advancements in intrusion detection systems for the internet of things," *Security and Communication Networks*, vol. 2019, Article ID 4301409, 19 pages, 2019.

[8] W. Khan, A. Daud, J. A. Nasir, and T. Amjad, "A survey on the state-of-the-art machine learning models in the context of NLP," *Kuwait journal of Science*, vol. 43, no. 4, 2016.

[9] Z. Hu, G. Xu, and X. Zheng, "SSL-SVD: semi-supervised learning-based sparse trust recommendation," *ACM Transactions on Internet Technology*, vol. 20, no. 1, pp. 1–20, 2020.

[10] R. Chapaneri and S. Shah, "A comprehensive survey of machine learning-based network intrusion detection," *Smart Intelligent Computing and Applications*, vol. 1, pp. 345–356, 2019.

[11] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu, "A survey on machine learning techniques for cyber security in the last decade," *IEEE Access*, vol. 8, pp. 222310–222354, 2020.

[12] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhabad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 493–501, 2019.

[13] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, no. 1, pp. 949–961, 2019.

[14] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janiche, "Deep learning for cyber security intrusion detection: approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, Article ID 102419, 2020.

[15] A. Aldweesh, A. Derhab, and A. Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: a survey, taxonomy, and open issues," *Knowledge-Based Systems*, vol. 189, Article ID 105124, 2020.

[16] D. S. Vijayakumar and S. Ganapathy, "Machine learning approach to combat false alarms in wireless intrusion detection system," *Computer and Information Science*, vol. 11, no. 3, pp. 67–81, 2018.

[17] J. Man and G. Sun, "A residual learning-based network intrusion detection system," *Security and Communication Networks*, vol. 2021, Article ID 5593435, 9 pages, 2021.

[18] B. Riyaz and S. Ganapathy, "A deep learning approach for effective intrusion detection in wireless networks using CNN," *Soft Computing*, vol. 24, pp. 17265–17278, 2020.

[19] C. Xu, J. Shen, and X. Du, "A method of few-shot network intrusion detection based on meta-learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3540–3552, 2020.

[20] W. Li, W. Meng, and M. H. Au, "Enhancing collaborative intrusion detection via disagreement-based semi-supervised learning in IoT environments," *Journal of Network and Computer Applications*, vol. 161, Article ID 102631, 2020.

[21] P. Nancy, S. Muthurajkumar, S. Ganapathy, S. V. N. Santhosh Kumar, M. Selvi, and A. Kannan, "Intrusion detection using dynamic feature selection and fuzzy temporal decision tree classification for wireless sensor networks," *IET Communications*, vol. 14, no. 5, pp. 888–895, 2020.

[22] C. M. Hsu, M. Z. Azhari, H. Y. Hsieh, S. W. Prakosa, and J. -S. Leu, "Robust network intrusion detection scheme using long-short term memory based convolutional neural networks," *Mobile Networks and Applications*, vol. 26, pp. 1–8, 2020.

[23] J. Gao, L. Gan, and F. Buschendorf, "Omni SCADA intrusion detection using deep learning algorithms," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 951–961, 2021.

[24] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, and Y. Huang, "Hast-I. D. S: learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[25] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modelling," 2018, https://arxiv.org/pdf/1803.01271.

[26] "Electronic railway equipment-train communication network— part 2-3: tcn communication profile, document IEC, 61375-61382-3," 2015.

[27] "Electronic railway equipment-train communication network— part 2-5: ethernet train backbone, document IEC, 61375," 2014.

[28] "Electronic railway equipment-train communication network— part 3-4: ethernet consist network, document IEC, 61375," 2014.

[29] H. Zhang, P. Cheng, L. Shi, and J. Chen, "Optimal DoS attack scheduling in wireless networked control system," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 3, pp. 843–852, 2015.

[30] Y. H. Tung, H. C. Wei, Y. W. Ti, and C. M. Yu, "Counteracting UDP flooding attacks in SDN," *Electronics*, vol. 9, no. 8, p. 1239, 2020.

[31] Z. Li, W. Xing, S. Khamaiseh, and D. Xu, "Detecting saturation attacks based on self-similarity of OpenFlow traffic," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 607–621, 2019.

[32] P. Anu and S. Vimala, "A survey on sniffing attacks on computer networks," in *Proceedings of the 2017 International Conference on Intelligent Computing and Control (I2C2)*, pp. 1–5, IEEE, Coimbatore, India, June 2017.

[33] W. Cao and X. Bao, "The research on the detection and defense method of the smurf-type DDos attack," in *Proceedings of International Conference on Soft Computing Techniques and Engineering Application*, pp. 315–324, Springer, New Delhi, India, July 2014.

[34] Y. Li, L. Zhu, H. Wang, F. R. Yu, and S. Liu, "A cross-layer defense scheme for edge intelligence-enabled CBTC systems against MitM attacks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2286–2298, 2020.

[35] A. H. Tahoun and M. Arafa, "Cooperative control for cyber–physical multi-agent networked control systems with unknown false data-injection and replay cyber-attacks," *ISA Transactions*, vol. 110, pp. 1–14, 2021.

[36] S. Sun, X. Fu, B. Luo, and X. Du, "Detecting and mitigating ARP attacks in SDN-based cloud environment," in *Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 659–664, IEEE, Toronto, Canada, July 2020.

[37] R. Little and D. Rubin, "Missing not at random models," in *Statistical Analysis With Missing Data*, John Wiley & Sons, Hoboken, NJ, USA, 3rd edition, 2019.

[38] J. Sim, J. S. Lee, and O. Kwon, "Missing values and optimal selection of an imputation method and classification algorithm to improve the accuracy of ubiquitous computing applications," *Mathematical problems in engineering*, vol. 2015, Article ID 538613, 14 pages, 2015.

[39] A. C. H. Choong and N. K. Lee, "Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method," in *Proceedings of the 2017 International Conference on Computer and Drone*

Applications (IConDA), pp. 60–65, IEEE, Kuching, Malaysia, November 2017.

[40] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks: a unified approach to action segmentation," in *Proceedings of the European Conference On Computer Vision (ECCV)*, pp. 47–54, Springer, Amsterdam, The Netherlands, October 2016.

[41] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *Proceedings of the IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*, pp. 156–165, Honolulu, HW, USA, July 2017.

[42] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings Of the IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*, pp. 3431–3440, Boston, MA, USA, June 2015.

[43] A. V. D. Oord, S. Dieleman, H. Zen et al., "Wavenet: a generative model for raw audio," 2016, https://arxiv.org/abs/1609.03499.

[44] K. He, X. Zhang, S. Ren et al., "Deep residual learning for image recognition," in *Proceedings Of the IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, June 2016.

[45] T. Salimans and D. P. Kingma, "Weight normalization: a simple reparameterization to accelerate training of deep neural networks," 2016, https://arxiv.org/abs/1602.07868.

[46] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift,," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2682–2690, Long Beach, CA, USA, June 2019.

[47] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings Of the International Conference On Machine Learning*, pp. 807–814, Haifa, Israel, June 2010.

[48] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: a survey," 2021, https://arxiv.org/abs/2102.04906.

[49] A. Graves, "Adaptive computation time for recurrent neural networks," 2016, https://arxiv.org/abs/1603.08983.

[50] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, https://arxiv.org/abs/1312.4400.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, https://arxiv.org/abs/1409.1556.

[52] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: an unsupervised intrusion detection system for high dimensional CAN bus data," *IEEE Access*, vol. 8, pp. 58194–58205, 2020.

[53] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "INDRA Intrusion detection using recurrent autoencoders in automotive embedded systems," 2020, https://arxiv.org/abs/2007.08795.

[54] S. Longari, D. H. N. Valcarcel, M. Zago, and M. Carminati, "CANnolo: an anomaly detection system based on LSTM autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, vol. 18, pp. 1913–1924, 2020.

[55] P. R. Kanna and P. Santhi, "Unified deep learning approach for efficient intrusion detection system using integrated spatial–temporal features," *Knowledge-Based Systems*, vol. 226, Article ID 107132, 2021.