Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.081193

ARTICLE

Exposing and Exploiting the Ineffectiveness of Real-Time ML-Based NIDSs on Ambiguous Traffic in Industrial IoT Networks

Mohammad Shamim Ahsan[1], Matthew G. Gaber[2], Mohiuddin Ahmed[3] and Al-Sakib Khan Pathan[4,*]

[1]College of Information Sciences and Technology, The Pennsylvania State University, State College, PA, USA
[2]School of ICT, SIIT Thammasat University, Pathum Thani, Thailand
[3]School of Computer Science and Information Technology, Adelaide University, Adelaide, Australia
[4]Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh

*Corresponding Author: Al-Sakib Khan Pathan. Email: sakib.pathan@gmail.com

Received: 25 February 2026; Accepted: 28 July 2026; Published: 22 October 2026

ABSTRACT: Pure TCP ACK packets, defined as acknowledgments without payload, are inherently ambiguous in network traffic classification. Their header fields are dictated entirely by TCP connection state, rather than application behavior, so field value variations do not contain behavior semantics that would reliably distinguish benign from malicious traffic. Such packets are prevalent in Industrial Internet of Things (IIoT) environments and occur at high volumes due to the communication characteristics of domain-specific protocols. These properties motivate an investigation into the effectiveness of Machine Learning Network Intrusion Detection Systems (ML-NIDSs) in IIoT networks for continuous monitoring of network activity. Our empirical evaluation demonstrates that state-of-the-art real-time detection models consistently misclassify pure ACK packets, across both benign and attack classes, with high confidence. To illustrate the security implications, we introduce a novel attack variant, the *DUP ACK Flooding attack*, which deliberately exploits this blind spot. We demonstrate its practical feasibility through testbed experiments under real-world conditions and analyze its impact on security monitoring and network operations. Finally, an ablation study in which pure ACK packets are withheld during training and introduced only at inference shows that the approach does not fully mitigate the issue, highlighting the need for further research.

KEYWORDS: Industrial IoT; intrusion detection system; machine learning; network security; ambiguous network traffic

1 Introduction

In practical applications such as an HTTP (Hypertext Transfer Protocol) session or bulk data transfer, a client typically initiates a request, and a server sends data in chunks (a.k.a., *segments*). In response to each downloaded data segment, the client confirms successful delivery by sending a pure acknowledgment packet ("pure TCP ACK") that does not contain any application payload. Such unidirectional data flows [1,2] are common in Industrial Internet of Things (IIoT) networks due to the domain-specific behaviors of widely used protocols, especially MQTT [3] (Message Queuing Telemetry Transport) and Modbus TCP/IP [4]. In particular, MQTT [3] is a client-server publish/subscribe protocol that provides various control packets, among which PUBLISH is the most predominant. IIoT devices (clients) publish their sensor readings to an MQTT server using PUBLISH control messages tagged with separate topics (distinct labels on the messages), and the server sends corresponding acknowledgment receipts. Subsequently, copies of these sensor data are delivered to subscribed clients (i.e., mobile device, backend system, and embedded system), who then

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

respond with pure TCP ACK packets for each successful delivery. Turning to Modbus TCP/IP, it is also an application-layer client/server (master/slave) protocol with a TCP interface that runs over Ethernet [4]. This protocol supports different functions, especially to READ or WRITE single/multiple registers, which are located in the memory of the Modbus server. Whenever a client initiates a Modbus request by explicitly specifying register addresses, the server responds accordingly by building a response based on the context, which is generally a pure ACK packet.

In IIoT networks, the core operation involves transmitting heterogeneous physical data from distributed sensors, which produces a high prevalence of pure TCP ACK packets. These are small, payload free control packets whose role is solely to ensure reliable and ordered data delivery. Because they carry no payload, their header fields, including sequence number, acknowledgment number, and flag bits, reflect only protocol mechanics rather than application behavior. Without context such as the volume, timing, or content of related packets, classifying a pure TCP ACK as benign or malicious is not reliably possible. We therefore designate them as *ambiguous traffic*.

This ambiguity can impact real-time intrusion detection systems that operate at the packet level. In these systems, features are extracted from each packet as it arrives and passed to a classifier without waiting for a flow to complete. This raises an open question: can these systems correctly classify pure TCP ACKs that originate from both benign and attack traffic?

To understand why this is non-trivial, it helps to compare two dominant feature extraction approaches [5,6]. Flow based features such as total bytes, flow duration, and packet length summaries, aggregate information across all packets in a connection and are effective at characterizing overall network behavior. However, they require the flow to be complete before classification is possible, making them better suited to offline or retrospective analysis [5,7]. This limitation is especially problematic for slow-rate attacks such as Slowloris, where the malicious pattern only emerges over an extended period. By the time a flow-based system could classify the traffic, damage may already be done. In contrast, packet based features are extracted independently from each packet on arrival, which enables near real-time detection without waiting for flow termination. However, the ambiguity of TCP ACKs is most exposed in packet based detection.

From these observations, we attempt to investigate the efficacy of the existing real-time machine learning-based NIDSs on ambiguous pure TCP ACK packets. To the best of our knowledge, this is the first work to date that provides an empirical evaluation of real-time ML-NIDSs, especially with ambiguous network traffic in IoT networks. Interestingly, we observe that state-of-the-art ML-NIDSs misclassify pure TCP ACK packets in a consistent manner (see Section 4). To understand the severity of such misclassifications, we explore how this blind spot can be exploited by an attacker. Accordingly, a new variant of existing cyberattack, referred to as **DUP ACK Flooding Attack**, is proposed to target the ML-NIDSs deployed in IIoT networks. In particular, a malicious insider overloads legitimate TCP sessions with constantly sending duplicate acknowledgments of the third packet (a pure ACK packet) of each TCP 3-way handshake. Although there are some similar ACK-related cyberattacks in the literature, such as ACK Flood DDoS attack [8,9] and DupACK spoofing attack [10], they differ from the proposed one in principle. As a mitigation strategy against such types of attack, we examine the performance of detection systems by retraining them without ambiguous pure TCP ACK packets. However, this technique does not appear to be an effective solution to prevent misclassifications. In summary, we make the following contributions:

- Five state-of-the-art real-time ML-NIDSs with 12 different models are investigated, and their ineffectiveness in correctly classifying ambiguous pure TCP ACK packets is empirically reported. Specifically, false negative rates (FNR) are excessively high at **24.78%–78.89%**. In addition, these advanced detection systems tend to exhibit high confidence even during misclassification.

- To explore the practical security implications of such misclassifications, we introduce a new variant of existing cyberattacks, called *DUP ACK Flooding attack*, targeting ML-NIDSs, and implement the demonstration in a controlled experimental setting. Furthermore, we analyze its impact on security operations center (SOC) activities and network performance.
- A defensive strategy that retrains models without pure ACK packets is explored, which results in a minimal change in the performance, except for one ML-NIDS, which is also due to the dominance of benign samples in the dataset.

2 Pure TCP ACK: Definition, Usage, and Structure

TCP (Transmission Control Protocol) divides every data stream into segments, where each segment represents a chunk of bytes. During transmission, a sender (S) sends a TCP segment. However, a sequence of segments can also be transmitted at once utilizing the *sliding window* mechanism [11]. To ensure reliable delivery, the corresponding packet of each segment contains a sequence number, representing the position of the byte stream of the data in that segment (i.e., the byte offset of its first byte). If the data bytes are successfully delivered, the receiver (R) acknowledges with a "*pure TCP ACK*" packet. A pure ACK network packet is defined as the TCP segment that does not contain any payload (application data), except that the ACK flag is set in the header and the options field (if present) may contain only the timestamp option used to measure the round-trip time [2]. To inform how many data bytes have been received, the ACK packet contains an acknowledgment number, indicating the offset of the byte that is expected to be received next. Importantly, the sequence number refers to the stream (S → R) flowing in the same direction as the segment (S → R), whereas the acknowledgment number represents the stream (S → R) flowing in the opposite direction of the segment (R → S). It is worth noting that although TCP supports full-duplex and piggybacked communication, it is irrelevant to our investigation. However, in real-world settings, duplicate acknowledgments (without any payload) are often observed in TCP communication due to packet loss or out-of-order delivery [11]. This mechanism facilitates handling any loss of data segment(s), especially the *fast retransmit* heuristic is triggered when three duplicate ACKs (except the original one) are received. These duplicate ACKs are also responsible for increasing the volume of pure TCP ACK traffic in IIoT networks.

A pure TCP ACK packet consists of a 14-byte Ethernet header, 20-byte IPv4 header, and 32-byte TCP header make each packet 66 bytes. Notably, the IP addresses, port numbers, checksum, sequence number, acknowledgment number, and window size fields vary across different pure TCP ACK packets; so their values are represented as X. The fragmentation bits are 010, signifying the packet must not be fragmented, since pure ACKs do not contain application payloads. All TCP flag bits, except ACK, are set to 0, indicating pure acknowledgment. In addition, the *options* field is flexible to 0–40 bytes, depending on its existence. However, for a pure ACK packet, only the timestamp option may be present (in that case, the packet length becomes 54 bytes); otherwise, the TCP options field remains empty. In this way, the length of a pure TCP ACK packet can be either 54 bytes or 66 bytes.

3 Experimental Methodology

3.1 Dataset Used

To perform our proposed empirical evaluation, two criteria must be satisfied. First, it must contain packet-based features, meaning that each row in the tabular formatted file represents the features of one packet (a.k.a, packet-level granularity). It should be noted that machine-learning-based network intrusion detection systems are commonly trained and evaluated using tabular datasets. Second, the dataset must closely replicate real-world industrial IoT settings and be widely accepted in the research community. Based on these criteria, we found that the *Edge-IIoTset* [12] dataset is best suited for our experiments, which

has become the most recognized benchmark for IIoT research in recent years, not only for representing practical industrial environments but also for its comprehensive coverage of the widest range of device types comprising 10 sensors, along with one of the 14 most recent sets of attacks. In addition to these, it has separate CSV files to train and test machine learning-based network intrusion detection systems. In total, the dedicated files contain 2,219,201 samples, among which 1,615,643 are of benign[1] traffic and 603,558 are of various attacks. It is worth mentioning that *no other* existing datasets meet our evaluation criteria.

Specific to our experiments, 685,029 packets are pure TCP ACK in the dataset, with 389,206 packets sent from client (IIoT device or attacker) to server (edge device) and 295,823 packets sent from server to client. The per-class packet counts are as follows: Normal (582,665), Backdoor (10,956), DDoS_HTTP (15,362), Password (23,124), Ransomware (4224), SQL_injection (23,865), Uploading (12,385), Vulnerability_scanner (5249), and XSS (7199). The percentage of pure ACK samples for each class is illustrated in Fig. 1. Overall, around 31% samples are pure TCP ACK across all classes, and it is not unusual for IIoT networks, as we already discussed (see Section 1).

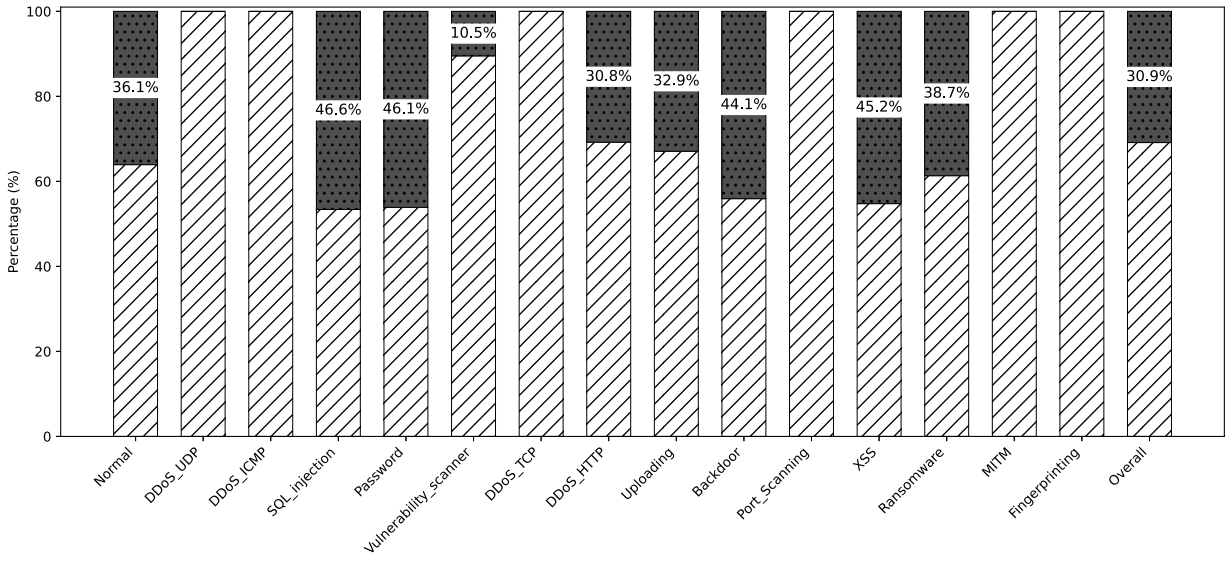

Figure 1: Distribution of pure ACK packets across different classes.

3.2 In-Depth Feature Analysis

The representative *Edge-IIotset* [12] dataset consists of 61 different packet-based attributes, covering various network, transport, and application layer protocols, including IP, HTTP, TCP, UDP, DNS, MQTT, and Modbus. Focusing on the TCP-oriented features, *tcp.ack*, *tcp.ack_raw*, *tcp.seq*, *tcp.che-cksum*, *tcp.len*, *tcp.flags*, *tcp.flags.ack*, *tcp.srcport*, and *tcp.dstport* are the relevant extracted features for pure TCP ACK packets. Based on characteristics, these features can be divided into two groups: *static* and *dynamic*. For pure ACK packets, *tcp.len*, *tcp.flags*, and *tcp.flags.ack* are static features. The value of *tcp.len* (TCP segment size) is always 0, as there is no payload or application data in pure acknowledgment. According to the protocol specification, the value of *tcp.flags* is 0×10, since only the ACK flag (*tcp.flags.ack*) is set to 1. For other fields-*tcp.ack*, *tcp.ack_raw*, *tcp.seq*, *tcp.checksum*, *tcp.srcport*, and *tcp.dstport*-values are varied. Thus, they are referred to as dynamic features. In particular, TCP sequence number ($tcp.seq$) is updated based on the equation $tcp.seq_i = tcp.seq_{i-1} + tcp.len_{i-1}$, where i and $i-1$ respectively denote the current and previous

[1]Since the dataset refers to the class of *benign* traffic as "Normal", we keep the same for consistency.

packets in the same direction. TCP acknowledgment number (*tcp.ack_raw*) can be calculated by adding the sequence number and the segment size of the previous packet from the opposite direction. However, it should be noted that *tcp.ack* and *tcp.seq* are relative values set by Wireshark (the packet analyzer tool); whereas *tcp.ack_raw* is the original acknowledgment number. Regarding the TCP checksum, it ensures the integrity of TCP packets by detecting errors during transmission. Specifically, at first, the sender calculates the checksum and includes it in the packet. Upon receipt, the receiver recalculates it and compares it to the one in the packet. This calculation relies on the TCP header, pseudo-header from the IP layer, and TCP payload, making its values dynamic.

The value distributions of each dynamic feature are illustrated using a violin plot (see Fig. 2). In particular, the order of classes in each subplot is sorted according to their feature median values. For *tcp.ack_raw*, the median and interquartile range (IQR) of the Normal class are comparatively smaller than the attack classes. In addition, the corresponding distribution is not as widely spread as the attacks. Moreover, the acknowledgment numbers related to the Ransomware and Backdoor attacks lack variation and are likely to center around a specific range. Regarding *tcp.ack* and *tcp.seq*, the values for most attack classes closely cluster, although slight variation is exhibited in the Normal class. Furthermore, nearly similar density distributions are observed in different classes for *tcp.checksum*, and no significant discrepancies are found for *tcp.srcport* and *tcp.dstport* features.

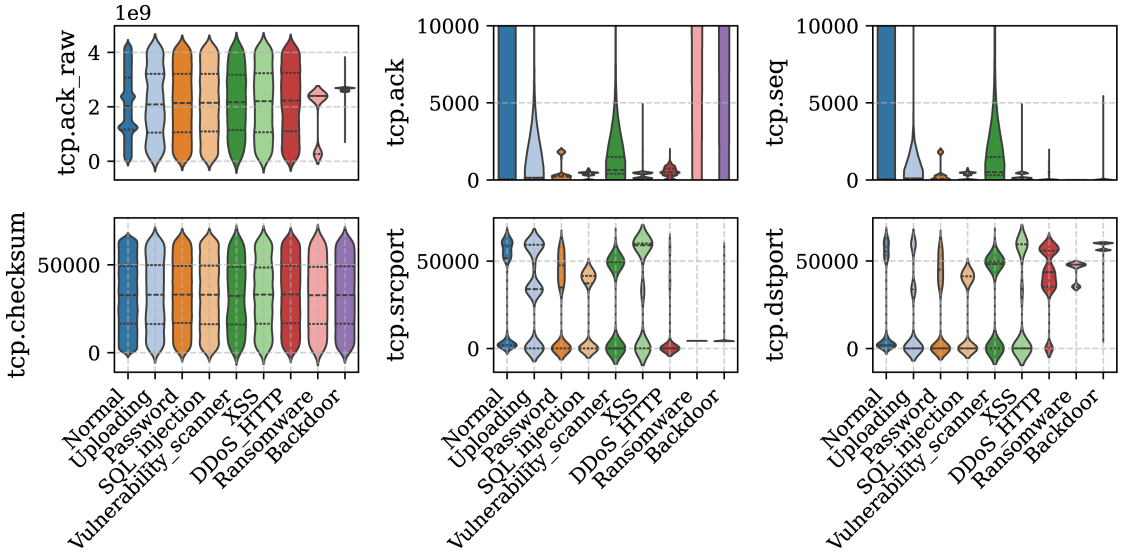

Figure 2: Distributions of the values of different TCP features related to pure ACK packets of normal and attack classes. To better visualize the difference, the *y*-axis of *tcp.ack* and *tcp.seq* has been limited to 10^4.

It is important to note that sequence and acknowledgment numbers may reveal temporal patterns in data transfer rates, burstiness, or round-trip time variations. However, the dataset used in this study does not contain the packet fields necessary to derive such temporal features. Moreover, such features are typically extracted from flow-level data, whereas our dataset comprises packet-level records. Consequently, the analysis done here is limited to the statistical features.

Observation: In the TCP protocol, for every new connection, an arbitrary sequence number, called the Initial Sequence Number (ISN), is generated and updated based on the data transfer in that session. In addition, the acknowledgment number changes with respect to the sequence number of the received packet and the corresponding data stream. Moreover, the source port, destination port, and checksum also depend on the use cases and clients, and can be changed seamlessly. Based on these observations, the deviations of

the outlined feature values across different classes, including Normal, do not carry any specific behaviors that can facilitate the distinction between whether the pure TCP ACK packets correspond to malicious traffic or just part of a legitimate operation.

4 Effectiveness of ML-NIDSs on Pure TCP ACKs

In this part, we experiment with the selected ML-NIDSs to evaluate their effectiveness on pure TCP ACK packets.

4.1 Selected Machine Learning-Based NIDSs

To examine the effectiveness of real-time detection systems, we select 12 representative models (five ML-NIDSs) that are publicly available, trained on the selected *Edge-IIoTset* dataset, well-regarded in academia, and demonstrate decent performance across all classes (see Table 1).

Table 1: Verification experiments of the selected ML-NIDSs.

ML-NIDS	Model	Acc. (%)	Prec. (%)	Rec. (%)	F1 (%)
ML-NIDS-1 [13]	DCNNBiLSTM	99.64	99.66	99.64	99.13
	Generic CNN	99.70	99.70	99.70	99.69
ML-NIDS-2 [14]	Xception	99.77	99.79	99.77	99.77
	VGG16	99.55	99.56	99.55	99.54
	VGG19	99.25	99.34	99.25	99.21
	Inception	99.77	99.79	99.77	99.78
	InceptionResNetV2	99.85	99.88	99.85	99.85
	EfficientNetB7	98.21	98.22	98.21	98.21
	EfficientNetV2L	99.78	99.79	99.78	99.77
ML-NIDS-3 [15]	CNN-LSTM-GRU	97	97	97	97
ML-NIDS-4 [16]	Hybrid CNN-LSTM	97.14	97.6≦	97.59≦	97.43≦
ML-NIDS-5 [17]	SF-CNN-LSTM	94.99	96.21	94.95	94.5

Note: ≦Although the authors explicitly reported the micro-average metrics, we include the weighted averages for consistency with the rest

The first detection system is DCNNBiLSTM [13], a popular NIDS for IoT in recent times, which utilizes a convolutional neural network (CNN) and Bidirectional LSTM (BiLSTM) networks. Since machine learning models require numerical values to train and test, one-hot encoding is applied to transform categorical features into numeric values during preprocessing. After that, the data are normalized using the Standardization technique to prevent the dominance of specific attributes. In addition, the Synthetic Minority Oversampling Technique (SMOTE) is used to reduce bias during training. The second system [14] employs Bootstrap ensembling to aggregate a diverse range of CNN models, specifically generic CNN, Xception, VGG16, VGG19, Inception, InceptionResNetV2, EfficientNetB7, and EfficientNetV2L. In this system, first, categorical features are scaled using Quantile normalization, then packets are transformed into images, and hyperparameters are tuned by the genetic algorithm (GA). The third ML-NIDS is also an ensemble classifier [15] comprising three advanced neural networks: CNN, LSTM, and GRU. Chi-squared test is applied to select the top 93 features that exhibit the most significant relationships with the attack classes (labels). Importantly, this detection system adopts two types of feature scaling techniques: label encoding to transform categorical strings into a numerical format, and one-hot encoding to convert categorical integer

features into a binary matrix to mitigate any potential misleading ordinal relationships among the numeric features. Although the last two systems integrate CNN and LSTM [16,17], the fourth model is trained solely on the transport and network layer features, and the fifth one is enhanced by statistical filtering techniques. Such diversity in similar neural networks will enable us to better examine real-world scenarios.

These NIDSs use an 80%–20% training-validation split. In addition, the built-in Python method "`train_test_split()`" was used with the "`stratify=y`" option to preserve the class distribution across both the training and validation sets, thereby reducing the risk of class imbalance affecting the evaluation results.

4.2 Metrics

A diverse range of performance metrics is adopted to evaluate the detection models. By definition, a true positive (TP) is the number of detected attack samples that occurred indeed, and a false positive (FP) is the cases where benign samples are misclassified as an attack. Likewise, true negative (TN) is the number of correctly predicted benign samples, and false negative (FN) is the volume of attack packets that are misclassified as benign. However, for multi-class classification, the definitions are slightly changed by extending their concepts. Consider a class, C_i. Now, TP_i is the number of C_i samples that are correctly classified as C_i, FP_i is the volume of other class samples that are incorrectly predicted as C_i, FN_i is the C_i sample count that are misclassified as other class, and TN_i is the number of non C_i samples that are classified as not C_i. However, the accuracy, precision, recall (or TPR), and F1 scores are formulated according to the standards. Notably, the weighted average of these metrics is calculated during the experiments to evaluate the overall effectiveness of ML-NIDSs. Imbalanced data distribution, where one class or a set of classes is dominant, is a common issue in publicly available IIoT/IoT datasets [18,19], including *Edge-IIoTset*. Taking this into account, we also measure the Cohen Kappa score (CKS), macro-average false positive rate (FPR), macro-average false negative rate (FNR), Matthew's correlation coefficient (MCC), and macro-average G-mean score. Their corresponding equations are as follows:

$$\mathrm{CKS} = \frac{p_o - p_e}{1 - p_e}, \quad \mathrm{FPR} = \frac{FP}{TN + FP}, \quad \mathrm{FNR} = \frac{FN}{TP + FN}$$

$$\mathrm{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TN + FN)(TN + FP)(TP + FN)(TP + FP)}} \quad \text{and} \quad \text{G-Mean} = \sqrt{TPR \cdot TNR}$$

where p_o is the observed agreement between two classifiers (or raters) and p_e is the expected agreement by chance. Notably, CKS is an inter-rater reliability statistic to quantify the percent of agreement between two raters in models' predictions [20].

4.3 Result Analysis

From Table 2, ML-NIDS-1 [13] and ML-NIDS-2 [14] significantly underperform on pure ACK packets, having an accuracy of 2.23%–39.66% and F1 score of 0.18%–30.28%. In addition, corresponding CKS, MCC, and G-mean are substantially low, sometimes even negative. The fifth ML-NIDS [17] performs slightly better than them; however, the results are still severe, as the accuracy, F1, and FNR are recorded at 71.70%, 76.79%, and 68.71%, respectively. Although the third [15] and fourth [16] detection models exhibit considerable accuracy, precision, recall, and F1, their false negative rates (FNR) are alarmingly high (24.78% and 32.21%, respectively). Based on prior knowledge of imbalanced data distribution, especially the dominance of the Normal class (85.06%) in the dataset, the incorrect prediction of benign pure ACKs could be considered as one of the possible reasons for such a high FNR. However, we have found that both models can correctly predict nearly all benign pure ACK packets, and misclassify attack packets not only as benign but also as

other attacks. Therefore, we further evaluate the systems considering only attack-class pure ACK samples. The investigation reveals a significant drop in all metrics, especially the accuracies reported as 67.65% and 78.81%, and F1 being 67.6% and 85.56%, respectively, suggesting that the models struggle to correctly classify attack samples.

Table 2: Performance of the ML-NIDSs on pure TCP ACK packets. All metrics, except the confidence score, are measured in percentage.

ML-NIDS	Model	Acc.	Prec.	Rec.	F1	CKS	FPR	FNR	MCC	G-Mean	Confidence⋆
ML-NIDS-1 [13]	DCNNBiLSTM	2.23	0.09	2.23	0.18	1.53	8.95	63.64	3.58	15.18	0.636/0.6135
ML-NIDS-2 [14]	Generic CNN	0	0	0	0	−5.72	9.72	72.73	−7.27	0	0.755/0.7274
	Xception	3.45	3.83	3.45	3.24	−11.39	10.25	71.02	−13.36	4.36	0.82/0.795
	VGG16	18.97	8.02	18.97	11.28	9.31	9.95	77.78	11.52	9.17	0.894/0.885
	VGG19	18.97	8.02	18.97	11.28	8.77	10.03	77.78	10.71	9.17	0.909/0.895
	Inception	39.66	38.47	39.66	30.28	28.60	6.36	53.21	31.71	21.69	0.574/0.54
	InceptionResNetV2	8.62	1.53	8.62	2.60	−2.81	10.28	73.75	−4.42	3.54	0.646/0.612
	EfficientNetB7	27.59	13.54	27.59	17.44	19.87	8.83	62.96	22.93	25.90	0.891/0.875
	EfficientNetV2L	1.72	15.52	1.72	3.10	1.46	9.83	78.89	5.03	3.33	0.663/0.629
ML-NIDS-3 [15]	CNN-LSTM-GRU	95.17	95.99	95.17	95.16	82.28	0.55	24.78	82.40	85.26	0.47/0.459
	(only attack packets)	67.65	73.19	67.65	67.6	61.19	4.99	27.87	62.5	81.46	0.47/0.459
ML-NIDS-4 [16]	Hybrid CNN-LSTM	97.45	97.36	97.45	97.28	87.32	1.91	32.21	87.76	76.01	0.847/0.843
	(only attack packets)	78.81	94.26	78.81	85.56	57.09	2.95	32.21	60.67	65.39	0.847/0.843
ML-NIDS-5 [17]	SF-CNN-LSTM	71.70	85.24	71.70	76.79	30.09	3.19	68.71	34.40	30.59	0.956/0.951

Note: ⋆ Confidence scores are measured in unit scale; Values shown as x/y indicate the standard mean and geometric mean, respectively.

Confidence Score Analysis. Typically, a neural network uses a softmax function to generate final raw predictions for a multiclass classification task. In the literature, the maximum class probability obtained from this softmax output for each prediction is defined as the *confidence score* [21]. In this work, we utilize this metric to examine the confidence levels of the selected models during misclassification. The experimental results are depicted in the last column of Table 2. Regarding poor-performing detection systems, the second ML-NIDS models based on generic CNN, Xception, VGG16, VGG19, EfficientNetB7, and the fifth NIDS exhibit high confidence (0.755–0.998) during the misclassification of pure ACK packets. In contrast, the first ML-NIDS and three models from ML-NIDS-2: Inception, InceptionResNetV2, and EfficientNetV2L are found to be moderately confident about incorrect predictions. Interestingly, the third detection system shows low confidence (around 0.47), although other performance metrics are high. Further investigation of its scores reveals that this model is completely confident during (correct) classification of pure ACK packets from the Normal class, but under-confident about the mispredicted packets of attack classes. However, the confidence score of the relatively better-performing model, ML-NIDS-4, is higher for both Normal (correct predictions) and attack classes (incorrect predictions). In general, it can be concluded that most detection models confidently misclassify pure ACK samples, and all models fail to reliably classify attack class samples.

Takeaway: The significant performance degradation on pure TCP ACK packets indicates that existing real-time machine learning-based NIDSs are not effective enough for real-world deployment. It also signifies that any external attacker or malicious insider can exploit the blind spots of these ML-NIDSs to disrupt the normal workflow of the security operations center (SOC). For example, simply sending pure acknowledgments to the network can trigger *false alarms* in the system, since benign pure ACKs are often misclassified as attacks (see Section 5). In addition, the incorrect classification of attack packets as another

attack can mislead security analysts and severely affect recovery actions, resulting in delayed responses to the genuine attack. Eventually, the system becomes more vulnerable, sensitive, and untrustworthy.

5 Proposed DUP ACK Flooding Attack

From the findings that existing state-of-the-art NIDSs misclassify "benign" pure TCP ACK packets as "attack", a malicious insider—who has legitimate access to the system—can exploit this blind spot of ML-NIDSs. An overview of how an attacker can leverage these misclassifications is illustrated in Fig. 3. In particular, the attacker can *intentionally* send multiple pure DUP ACKs to the server to overwhelm the system, eventually overloading RAM and CPU usage. In addition, these false positive packets misclassified by ML-NIDSs will trigger (unexpected) false alarms in the system, which can significantly delay the response of the genuine attack alerts as well as waste a large number of computational resources and manpower of the Security Operations Center (SOC). An experimental validation of such cyberattacks, refer to as ***DUP ACK Flooding attack***, is provided in the next subsection. Since the device is authorized to the system, these packets are legitimate and are not blocked or dropped by the intermediate routers or the server's TCP/IP stack.

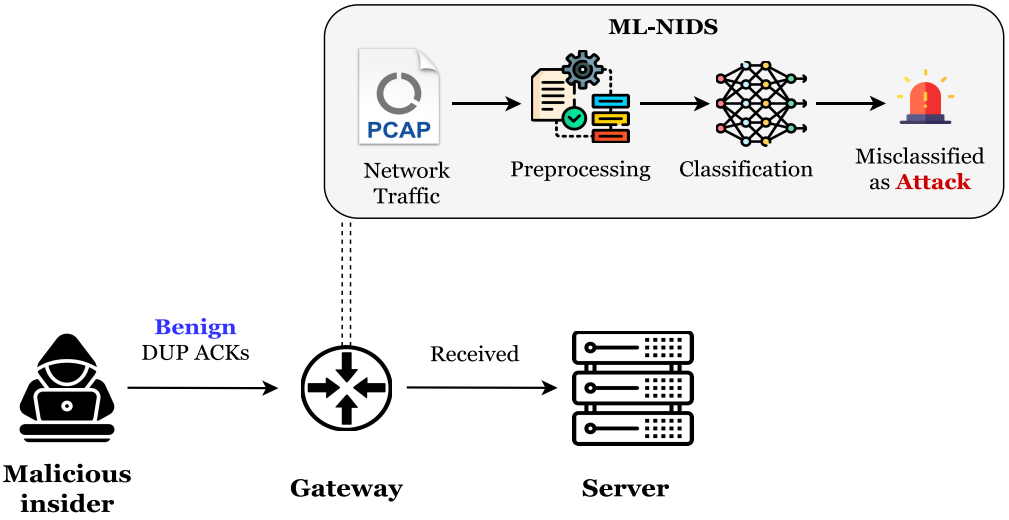

Figure 3: Overview of the proposed exploitation.

5.1 Demonstration of the DUP ACK Flooding Attack

Recall the usage of the ACK packet in TCP communication: in most cases, a pure TCP ACK packet is sent by the receiver as a confirmation of the data segment received recently. In addition, the last (3rd) packet of TCP connection establishment is also a pure TCP ACK packet. Regrading duplicate ACKs, these are generated to handle packet losses and recover out-of-order deliveries. Specifically, when loss of any data segments is observed at the receiver's end, it sends three duplicate ACKs. From a sender's perspective, packet loss means that multiple acknowledgments will arrive, each carrying the same sequence number. After a series of three duplicates plus an original ACK arrives, the *fast retransmit* heuristic triggers a retransmission without further waiting [11].

However, *what if there is no data loss, but the receiver consistently sends duplicate ACKs with no payload?* Two conditions need to be satisfied here: (C1) DUP ACKs can not contain but headers, and (C2) they must not be the confirmation of any received application data segments, ensuring no data loss has occurred in the system. Maintaining C1 is not an arduous task, since such packets can be easily crafted. However, the most crucial point here is to find a legal way to send ACKs that are not a response to any received data segments.

For this purpose, a plausible strategy can be leveraging the TCP 3-way handshake, which is used during a session establishment. In particular, the standard mechanism works as follows: (i) the initiator (client) sends a SYN packet to the server; (ii) the server responds with a SYN-ACK packet, acknowledging the client's request and advertising its own initial sequence number; and (iii) the client confirms receipt by sending a final (pure) ACK packet, as illustrated in Fig. 4. Since the last packet is pure ACK and does neither have any payload nor act as a receipt of application data segments, this packet should be duplicated in theory. This is the concept of our proposed ***DUP ACK Flooding attack***. Specifically, an attacker can exploit legitimate TCP sessions and utilize the following two simple sequential steps to carry out this attack.

1. Establish a standard TCP connection using a 3-way handshake
2. After a successful connection establishment, repeatedly transmit duplicate ACK packets (duplication of the last packet of the handshake) without changing the sequence and acknowledgment numbers.

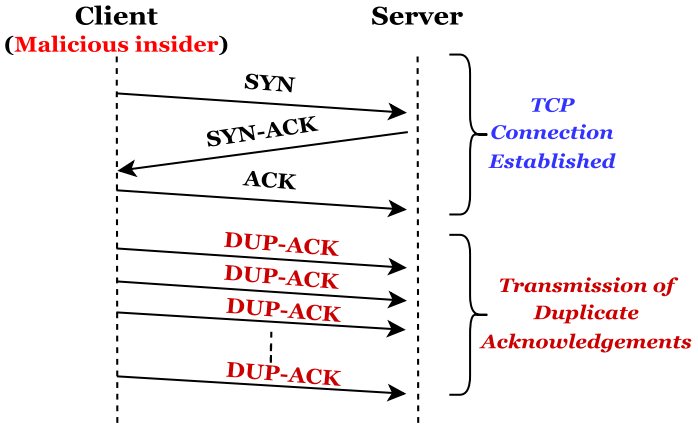

Figure 4: Overview of the DUP ACK flooding attack.

This attack is functional, as the crafted DUP ACK packets do not violate any router or firewall policies and seamlessly go through them. Moreover, since these DUP ACKs do not represent any application data segments, the sender (server) does not have anything to retransmit; thus, its TCP/IP stack simply remains in the same state and does nothing, implying that these packets are **not rejected** at the server's end.

Experimental Setup. To evaluate the real-world feasibility of the DUP ACK Flooding attack, we conduct a testbed experiment in a controlled laboratory environment. The simulation environment is Mininet VM [22], running Ubuntu 20.04.1 LTS with Linux kernel version 5.4.0-42-generic. This is an emulator that replicates a complete network of hosts, links, and switches on a single machine. Mininet hosts and switches use the Linux network stack, allowing realistic packet-level behavior while maintaining full experimental control. We emulate a simple three-node topology consisting of a sender host (`h1`), a router (`r1`), and a receiver host (`h2`) arranged in a linear path, as illustrated in Fig. 5. Hosts `h1` and `h2` reside in the subnets 10.0.0.0/24 and 10.0.1.0/24, respectively, with corresponding interfaces 10.0.0.1/24 (`h1-eth0`) and 10.0.1.2/24 (`h2-eth0`). These hosts act as an attacker and a server in our experiment and are connected through an intermediate router `r1`, which contains interfaces `r1-eth0` (10.0.0.254/24) and `r1-eth1` (10.0.1.254/24) to serve as the gateway for the subnets, with IP forwarding enabled to allow smooth communication between them.

Attack Procedure. At the beginning of the attack, `h1` (malicious insider) initiates a TCP connection by sending a SYN packet. The server host `h2` replies with a SYN-ACK, followed by an ACK packet from the attacker. After successfully establishing the session, `h1` repeatedly transmits DUP ACK packets without

changing the sequence and acknowledgment numbers of the last ACK packet. This process lasts for 30 s for each session. To generate the crafted traffic, a Python library, `Scapy`, is used. For this experiment, the attacker (`h1`) creates three TCP sessions; however, in real-world settings, this number can be much higher. More concerningly, when using a botnet, the number of connections–and consequently the number of DUP ACK packets–can be massive. Regarding data collection, traffic is captured at two distinct endpoints. First, packets are collected at the router interface `r1-eth0` to verify that the DUP ACK packets successfully pass through the router. Since the attack packets are not dropped, it validates that the crafted DUP ACK packets do not violate network-level or protocol-level constraints. Second, traffic is captured at the server endpoint (`h2-eth0`) to examine whether the packets are rejected by the TCP/IP stack. No packet rejection is observed at this stage, indicating that DUP ACK packets reach the server without intermediate intervention and are capable of producing the intended effects. The resulting `pcap` files are publicly available and can be accessed from the Git repository[2].

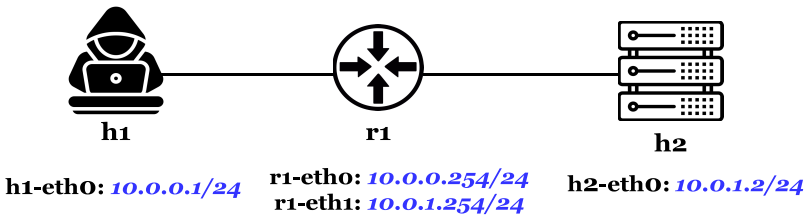

Figure 5: Experimental network topology.

It is important to note that modern real-time NIDSs often rely on time-windowed flow statistics (e.g., analyzing traffic over intervals of a few seconds). As a result, injecting DUP ACKs under current conditions would likely be detected by such window-based rate monitoring. However, intuitively, these deliberately duplicated ACKs can also be injected later during ongoing data transmission. Since the packets are valid duplicates of previously observed ACKs, transmitting them does not violate TCP/IP protocol rules and would not typically be rejected by routers or firewalls. Instead, the receiver's TCP stack would generally remain in the same state and continue processing incoming packets as usual. Another worth mentioning point is that traditional, stateful firewalls or signature-based IDSs (e.g., Snort, Suricata) could detect the continuous flooding of ACK packets. However, as our focus is on ML-based NIDSs, other types of IDSs are beyond the scope of this work. Additionally, as discussed in the preceding paragraph, injecting duplicate ACKs during ongoing data transmission may also enable evasion of traditional signature-based NIDSs.

5.2 Attack Impacts

The proposed DUP ACK Flooding attack can severely damage both the monitoring of the security operations center (SOC) and the operations of a server, as discussed below.

Security Monitoring Impacts. The misclassification of pure ACK packets by state-of-the-art ML-NIDSs will trigger *false positive alerts* in the system, which can significantly affect the normal workflow of a security operations center (SOC). Such incidents typically lead to two critical consequences. First, computational resources and human effort are wasted, leading to higher operational overhead and less efficient incident response. Second, the proliferation of false alarms hampers investigations of genuine threats to some extent, ultimately increasing the network's exposure to real attacks. According to IBM's survey [23] conducted in 2023, false positives are listed as one of the top four challenges for SOC, and 76% of analysts blame false

[2] https://github.com/mohammad-shamim-ahsan/Empirical-Study-IIoT.

alerts as the cause of delayed response to real threats. In addition, 81% of SOC teams report that manual investigation of threats significantly slows down their overall response times.

Operational Overheads. Since the crafted DUP ACK packets are accepted by the server, even though the server does not alter the TCP protocol state or generate any response, it is still required to process every incoming packet across multiple parallel sessions. For every such packet, the server performs packet parsing, checksum verification, and connection state lookup, temporarily allocating buffers that are released immediately after processing, which can lead to increased CPU utilization. Moreover, as a side effect of the flooding attack, these attack packets may consume a substantial portion of the server's available network bandwidth, thereby degrading the server's ability to handle normal traffic. It is worth noting that these effects could be massive if launched from a network of compromised devices—a botnet.

It should be noted that measuring CPU utilization and memory overhead, as well as comparing the proposed attack with existing attacks in terms of resource efficiency, could further strengthen the practical relevance of this work. However, the proposed attack introduced only a marginal additional workload in our experimental setup, resulting in negligible resource overhead. Consequently, CPU utilization and memory consumption showed no sufficiently distinguishable patterns to enable a meaningful comparison of resource efficiency among different attacks.

5.3 Distinction with Existing Attacks

The closest existing variant of DUP ACK Flooding attack is *ACK Flood DDoS attack* [8,9]. Although they may seem similar at first glance, they substantially differ in principle. An ACK Flood DDoS attack aims to bypass TCP connection establishment by sending a large number of TCP ACK packets to the target system without any previous handshake. These ACK packets do not contain any payload and are frequently generated by a botnet containing different IP addresses. Importantly, these spoofed ACK packets, which appear legitimate but do not belong to any established TCP session, will eventually be dropped by the receiver's TCP/IP stack, since they are not a part of any existing connection. In contrast, because our crafted DUP ACK packets are part of a valid TCP session, they are accepted by the receiver. As a consequence, it becomes more powerful than a conventional ACK flood–based DDoS attack. Another similar attack is *DupACK spoofing attack* [10], which exploits TCP's inherent inability to distinguish a valid duplicate ACK from a spoofed one. Specifically, after receiving a data segment, the attacker forges a long stream of DUP ACKs that acknowledge the last sequence number received. As a consequence, the fast retransmit heuristic is triggered at the sender's end, which retransmits the missing segment and subsequently sends additional new segments in response to each further forged duplicate ACK. On the other hand, the DUP ACK flooding attack does not leverage acknowledgments corresponding to any previously sent or in-flight data segment, and therefore does not induce the sender to transmit new data.

6 Ablation Study: A Possible Mitigation Strategy

To prevent the proposed cyberattack introduced to exploit the blind spot of ML-NIDSs, one trivial method could be to retrain the models without pure TCP ACK packets and consider these packets as "unseen" samples during inference. Since neural networks, especially CNNs and LSTMs, are well known for their generalization ability [24] and pure TCP ACK packets do not contain any exceptional network features (see Section 2), this approach appears justifiable.

Table 3 shows the performance of the retrained models on pure ACKs. As shown in the table, retraining does not improve the performance of the first and second ML-NIDSs. Regarding the third NIDS, a slight decrease (<3%) is observed in relevant metrics. However, ML-NIDS-5 exhibits a remarkable jump, especially with an increase of 12.64%, 0.07%, 12.64%, 7.45%, 14.01% in the accuracy, precision, recall, F1, and MCC,

respectively. In addition, FPR and FNR are also reduced ($\Delta = -1.75\%$ and $\Delta = -6.03\%$). However, the G-mean score is decreased (30.59% → 20.7%), which indicates a reduced TNR, considering the increase in TPR (recall). Moreover, retraining of the fourth ML-NIDS results in low training accuracy (~40.34%), likely due to the limited features (18) used, and is therefore excluded from the experiment. To examine whether retraining brings about any unexpected side effects, our investigation proceeds with a test set that does not contain any pure TCP ACK packets. From Table 4, a minimal increase (0.2%–0.5%) is observed in the metrics for the third and fifth ML-NIDSs. In addition, Xception, Inception, and InceptionResnetV2 models of the second detection system also exhibit a decent improvement of roughly 2%–3%. However, the performance of other models of this NIDS and the first ML-NIDS has slightly declined. Taking both experiments into account, the adopted approach for retraining *appears to* improve the ML-NIDS-5 [17], although it does not substantially influence other machine learning-based NIDSs.

Next, we conduct an in-depth analysis of ML-NIDS-5 and compute its per-class precision, recall, and F1 scores. According to Table 5, substantial improvement is reported in the Normal class metrics, indicating that the retrained model can better detect benign pure ACK samples; specifically, 89,848 more correct predictions than before. Regarding the Backdoor samples, no significant changes are observed in the statistics. However, the performance on Ransomware packets is substantially decreased, which is reflected in the recall (100% → 13%). Among 4224 pure ACKs of this class, the retrained model can correctly classify only 563 samples. However, packets from other classes still remain undetected. Notably, improvement in benign pure ACK packets has a global impact on overall performance metrics, since the Normal class is dominant in the representative dataset. Another interesting point to note is that the retrained model mispredicts some pure ACKs as the DDoS_ICMP and Port_Scanning attacks, which in fact do not contain any pure ACK packets. In contrast, the original model classifies all pure ACKs, whether correctly or incorrectly, into the classes that actually contain this type of packet.

Takeaway: Retraining intrusion detection models without ambiguous pure ACK packets has negligible impacts, except for one model that demonstrates considerably better ability to correctly classify benign pure TCP ACK packets. However, the main reason behind this improvement is likely be the high prevalence of benign traffic in the dataset, which increases the probability of randomly classifying an *unknown* pure ACK sample as benign. Overall, this investigation reveals that considering all pure ACKs as independent packets is not an effective solution in general.

Table 3: Performance of the retrained ML-NIDSs on pure TCP ACK packets. Grey-colored cells indicate a statistical decrease in the metrics, whereas the green-colored cells represent a statistical increase.

ML-NIDS	Model	Acc.	Δ	Prec.	Δ	Rec.	Δ	F1	Δ	CKS	Δ	FPR	Δ	FNR	Δ	MCC	Δ	G-Mean	Δ
ML-NIDS-1 [13]	DCNNBiLSTM	1.75	−0.48	0.17	0.08	1.75	−0.48	0.31	0.13	0.23	−1.3	8.32	−0.63	64.71	1.07	0.46	−3.12	11.55	−3.63
ML-NIDS-2 [14]	Generic CNN	0	0	0	0	0	0	0	0	0	5.72	0	−9.72	10	−62.73	80	87.27	0	0
	Xception	0	−3.45	0	−3.83	0	−3.45	0	−3.24	−13.61	−2.22	10.49	0.24	72.73	1.71	−16.45	−3.09	0	−4.36
	VGG16	0	−18.97	0	−8.02	0	−18.97	0	−11.28	−12.89	−22.2	12.58	2.63	88.89	11.11	−23.24	−34.76	0	−9.17
	VGG19	0	−18.97	0	−8.02	0	−18.97	0	−11.28	−8.24	−17.01	10.88	0.85	80	2.22	−10.78	−21.49	0	−9.17
	Inception	29.31	−10.35	14.33	−24.14	29.31	−10.35	18.64	−11.64	19.31	−9.29	7.2	0.84	55.94	2.73	22.34	−9.37	14.73	−6.96
	InceptionResNetV2	0	−8.62	0	−1.53	0	−8.62	0	−2.6	−5.16	−2.35	9.6	−0.68	72.73	−1.02	−6.86	−2.44	0	−3.54
	EfficientNetB7	20.69	−6.9	5.09	−8.45	20.69	−6.9	8.16	−9.28	13.09	−6.78	8.67	−0.16	60	−2.96	16.47	−6.46	16.05	−9.85
	EfficientNetV2L	0	−1.72	0	−15.52	0	−1.72	0	−3.1	0	−1.46	11.1	1.27	88.8	9.91	0	−5.03	0	−3.33
ML-NIDS-3 [15]	CNN-LSTM-GRU	94.42	−0.75	96.03	0.04	94.42	−0.75	94.01	−1.15	79.5	−2.78	0.52	−0.03	27.87	3.09	79.79	−2.61	64.57	−20.69
ML-NIDS-4 [16]	Hybrid CNN-LSTM	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
ML-NIDS-5 [17]	SF-CNN-LSTM	84.34	12.64	85.31	0.07	84.34	12.64	84.24	7.45	46.82	16.73	1.44	−1.75	62.68	−6.03	48.41	14.01	20.77	−9.82

Table 4: Performance comparison of the ML-NIDSs on the packets excluding pure ACKs. Increment and decrement in the metrics are highlighted using blue and red colors, respectively.

ML-NIDS	Model	Before (%)				After (%)			
		Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
ML-NIDS-1 [13]	DCNNBiLSTM	95.66	96.45	95.66	95.37	94.38	93.97	94.38	93.51
ML-NIDS-2 [14]	Generic CNN	98.69	98.89	98.69	98.71	96.41	94.57	96.41	95.26
	Xception	97.71	99.31	97.71	97.44	99.35	99.61	99.35	99.38
	VGG16	99.35	99.42	99.35	99.36	97.71	96.54	97.71	96.86
	VGG19	96.08	94.89	96.08	95.01	90.20	85.54	90.20	87.05
	Inception	97.71	98.16	97.71	97.79	98.69	97.99	98.69	98.27
	InceptionResNetV2	97.39	97.98	97.39	97.27	99.67	99.69	99.67	99.67
	EfficientNetB7	98.37	97.71	98.37	97.95	92.48	88.42	92.48	89.86
	EfficientNetV2L	94.77	97.23	94.77	94.11	97.39	96.86	97.39	96.89
ML-NIDS-3 [15]	CNN-LSTM-GRU	97.59	97.79	97.59	97.57	97.81	98.24	97.81	97.75
ML-NIDS-4 [16]	Hybrid CNN-LSTM	97.97	97.99	97.97	97.76	—	—	—	—
ML-NIDS-5 [17]	SF-CNN-LSTM	95.74	96.47	95.74	95.50	95.94	96.74	95.94	95.76

Table 5: Per-class performances of ML-NIDS-5 on pure ACKs.

Class	Before (%)			After (%)			Support
	Prec.	Rec.	F1	Prec.	Rec.	F1	
Backdoor	7	100	13	12	100	21	10,956
DDoS_HTTP	0	0	0	0	0	0	15,361
Normal	100	82	90	100	97	99	580,057
Password	0	0	0	0	0	0	23,117
Ransomware	8	100	15	7	13	9	4224
SQL_injection	0	0	0	0	0	0	23,692
Uploading	0	0	0	0	0	0	12,063
Vulnerability_scanner	0	0	0	0	0	0	5225
XSS	0	0	0	0	0	0	7103

7 Conclusion

This paper presents an empirical study of the effectiveness of real-time machine learning-based NIDSs on pure TCP ACK packets. We found that state-of-the-art detection models, deployed in IIoT networks, consistently misclassify these ambiguous traffic samples. This reveals a potential blind spot in these CNN, LSTM, BiLSTM, GRU, DNN, and their integration-based detection systems. To practically demonstrate the real-world severity of these misclassifications, a new form of cyberattack has been proposed, where duplicate ACKs are sent to the server in a legitimate way to affect the systems. Finally, to alleviate such a drastic deficiency, we have examined retraining the ML-NIDS without pure ACK samples and observed minimal improvement in the performance of the models.

Since no other existing dataset meets our empirical evaluation criteria, we rely solely on the benchmark IIoT dataset, *Edge-IIoTset*. This reliance may raise concerns regarding the generalizability of our observations and the proposed attack. We recall the core observation of this work: the header fields of pure TCP ACK packets primarily encode TCP connection state rather than application-level behavior; therefore, variations in these field values do not convey meaningful behavioral semantics that can reliably distinguish benign from malicious traffic. On this basis, theoretically, ML models trained on packet-based features are likely to misclassify TCP ACKs across different classes.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Mohammad Shamim Ahsan; writing—original draft preparation, Mohammad Shamim Ahsan and Al-Sakib Khan Pathan; writing—review and editing, Mohammad Shamim Ahsan, Al-Sakib Khan Pathan, Mohiuddin Ahmed and Matthew G. Gaber. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available at https://github.com/mohammad-shamim-ahsan/Empirical-Study-IIoT.

Ethics Approval: Not applicable.

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

References

1. Li T, Zheng K, Xu K, Jadhav RA, Xiong T, Winstein K, et al. Revisiting acknowledgment mechanism for transport control: modeling, analysis, and implementation. IEEE/ACM Trans Netw. 2021;29(6):2678–92. doi:10.1109/tnet.2021.3101011.
2. Chan M, Cheriton DR. Improving server application performance via pure TCP ACK receive optimization. In: Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC 13); 2013 Jun 26–28; San Jose, CA, USA. p. 359–64.
3. OASIS Standard. MQTT Version 5.0 [Online]. 2019 [cited 2026 Feb 24]. Available from: http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf.
4. Acromag Incorporated. Introduction to Modbus TCP [Online]. 2005 [cited 2026 Feb 24]. Available from: https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf.
5. He K, Kim DD, Asghar MR. Adversarial machine learning for network intrusion detection systems: a comprehensive survey. IEEE Commun Surv Tutor. 2023;25(1):538–66. doi:10.1109/comst.2022.3233793.
6. Yang B, Arshad MH, Zhao Q. Packet-level and flow-level network intrusion detection based on reinforcement learning and adversarial training. Algorithms. 2022;15(12):453. doi:10.3390/a15120453.
7. Borgioli N, Thi Xuan Phan L, Aromolo F, Biondi A, Buttazzo G. Real-time packet-based intrusion detection on edge devices. In: Proceedings of Cyber-Physical Systems and Internet of Things Week 2023; 2023 May 9–12; San Antonio, TX, USA. p. 234–40.
8. John A. ACK flood DDoS attack: how to protect your website [Online]; 2022 [cited 2026 Feb 24]. Available from: https://arzhost.com/blogs/ack-flood-ddos-attack-types-of-ddos-attacks/.
9. Al-Naeem MA. Prediction of re-occurrences of spoofed ACK packets sent to deflate a target wireless sensor network node by DDOS. IEEE Access. 2021;9:87070–8. doi:10.1109/access.2021.3089683.
10. Savage S, Cardwell N, Wetherall D, Anderson T. TCP congestion control with a misbehaving receiver. ACM SIGCOMM Comput Commun Rev. 1999;29(5):71–8. doi:10.1145/505696.505704.
11. Comer DE. Chapter 11: reliable stream transport service (TCP). In: Internetworking with TCP/IP. Vol. 1. 6th ed. Boston, MA, USA: Addison-Wesley Professional; 2013. p. 199–243.

12. Ferrag MA, Friha O, Hamouda D, Maglaras L, Janicke H. Edge-IIoTset: a new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. IEEE Access. 2022;10:40281–306. doi:10.1109/access.2022.3165809.
13. Hnamte V, Hussain J. DCNNBiLSTM: an efficient hybrid deep learning-based intrusion detection system. Telemat Inform Rep. 2023;10:100053.
14. Latif S, Boulila W, Koubaa A, Zou Z, Ahmad J. DTL-IDS: an optimized intrusion detection framework using deep transfer learning and genetic algorithm. J Netw Comput Appl. 2024;221:103784.
15. Kilichev D, Turimov D, Kim W. Next-generation intrusion detection for IoT EVCS: integrating CNN, LSTM, and GRU models. Mathematics. 2024;12(4):571.
16. de Elias EM, Carriel VS, De Oliveira GW, Dos Santos AL, Nogueira M, Junior RH, et al. A hybrid CNN-LSTM model for IIoT edge privacy-aware intrusion detection. In: Proceedings of the 2022 IEEE Latin-American Conference on Communications (LATINCOM); 2022 Nov 30–Dec 2; Rio de Janeiro, Brazil. p. 1–6.
17. Imani F, Kargar M, Assadzadeh A, Bayani A. Integrating CNN-LSTM networks with statistical filtering techniques for intelligent IoT intrusion detection. In: Proceedings of the 2024 8th International Conference on Smart Cities, Internet of Things and Applications (SCIoT); 2024 May 14–15; Mashhad, Iran. p. 189–95.
18. Ahsan MS, Islam S, Shatabda S. A systematic review of metaheuristics-based and machine learning-driven intrusion detection systems in IoT. Swarm Evol Comput. 2025;96(22):101984. doi:10.1016/j.swevo.2025.101984.
19. Zhu Q. On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset. Pattern Recognit Lett. 2020;136(1):71–80. doi:10.1016/j.patrec.2020.03.030.
20. Chicco D, Warrens MJ, Jurman G. The Matthews correlation coefficient (MCC) is more informative than Cohen's Kappa and Brier score in binary classification assessment. IEEE Access. 2021;9:78368–81. doi:10.1109/access.2021.3084050.
21. Pearce T, Brintrup A, Zhu J. Understanding softmax confidence and uncertainty. arXiv:2106.04972. 2021. doi:10.48550/arXiv.2106.04972.
22. Lantz B, Heller B, McKeown N. A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks; 2010 Oct 20–21; Monterey, CA, USA. p. 1–6.
23. Consult M. Global security operations center study results [Online]; 2023 [cited 2026 Feb 24]. Available from: https://www.ibm.com/downloads/documents/us-en/10c31775a05401a5.
24. Younesi A, Ansari M, Fazli M, Ejlali A, Shafique M, Henkel J. A comprehensive survey of convolutions in deep learning: applications, challenges, and future trends. IEEE Access. 2024;12(2):41180–218. doi:10.1109/access.2024.3376441.