Peer-to-Peer Networking and Applications (2023) 16:1286–1309
https://doi.org/10.1007/s12083-023-01477-x

Efficient time-delay attack detection based on node pruning and model fusion in IoT networks

Wenjie Zhao[1] · Yu Wang[2] · Wenbin Zhai[1] · Liang Liu[1] · Yulei Liu[1]

Received: 6 December 2022 / Accepted: 10 March 2023 / Published online: 30 March 2023

Abstract
IoT devices are vulnerable to various attacks because they are resource-limited. This paper introduces a novel type of attack called time-delay attack. The malicious nodes delay packet forwarding by extending the processing time of packets, thus affecting the performance and availability of the network. This attack is very stealthy and difficult to detect because it does not violate any communication protocol. To the best of our knowledge, how to detect the time-delay attack in IoT networks is still an open problem. We first propose a machine learning-based baseline algorithm to detect the time-delay attack. It models the system features of each node and the forwarding time of packets to detect whether a node is malicious or not. However, the baseline algorithm needs to detect all nodes in the network, which causes unnecessary resource consumption. Moreover, using a single model in the baseline algorithm does not have high robustness. To reduce the overhead and improve the detection performance, we design an efficient Detection algorithm based on Node pruning and Model fusion (DNM). DNM uses node pruning to filter out suspected nodes from all nodes. The suspected nodes are then detected according to a fusion model. We conduct experimental evaluations based on the Cooja network simulator. The experimental results show that baseline and DNM possess close to 90% accuracy, and DNM significantly outperforms other algorithms with an average F1-score of 0.85.

Keywords IoT network · Malicious node detection · Time-delay attack · Model fusion

1 Introduction

With the development of sensors, wireless networks, and embedded computing, the Internet of Things (IoT) industry is rapidly developing and expanding. As a widespread infrastructure, IoT has been widely used in several fields, such as smart homes [1], smart grids [2], environmental monitoring [3], national defense, and so on.

In IoT networks, the communication range of a single sensor device (or a node) is limited. When a node transmits packets to another node outside the communication range, it is necessary to rely on other nodes within the communication range for forwarding. Nodes can communicate with each other by utilizing various IoT technologies and protocols (e.g., 5G [4], WiFi [5], ZigBee [6]), forming a multi-hop IoT network. The multi-hop nature of IoT networks provides excellent flexibility in routing, and nodes acting as packet relays can find the optimal next hop through routing protocols, such as IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [7]. However, the nodes are usually resource-constrained. They are at risk of being attacked by attackers and hijacked as malicious nodes. Malicious nodes can perform deliberate actions such as dropping, tampering, and replaying packets during packet transmission, thus destroying the stability and availability of the networks [8, 9]. Therefore, how to ensure the security of packets during the routing process has become a key concern.

✉ Liang Liu
liangliu@nuaa.edu.cn

Wenjie Zhao
wenjiezhao@nuaa.edu.cn

Yu Wang
wangyu1503@126.com

Wenbin Zhai
wenbinzhai@nuaa.edu.cn

Yulei Liu
liu_yulei@nuaa.edu.cn

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, 29 Jiangjun RD, Nanjing 211106, Jiangsu, China

[2] The Fifth Electronics Research Institute of the Ministry of Industry and Information Technology, 78 West Zhucun RD, Guangzhou 511370, Guangdong, China

Motivation In this paper, we focus on internal attacks in IoT networks. The existing literature on routing security mainly focuses on detecting and defending against traditional cyber attacks, e.g., black hole attacks, gray hole attacks, wormhole attacks, and Sybil attacks [8–10]. However, time-delay attacks in IoT networks are less noticed. In fact, time-delay attacks can cause significant harm in many scenarios, such as industrial process monitoring and control, prognostics health management, fire or intrusion detection systems, and traffic accident prediction [11–13]. These scenarios heavily rely on the timely transmission of packets and are vulnerable to time-delay attacks.

Taking Fig. 1 as an example, it illustrates the differences in the forwarding behavior of benign and malicious nodes. Node N_1 delivers packet P_n to node N_{Sink}, which passes through $N_2, N_3, N_4, \ldots, N_t$. We assume that there is a malicious node N_3 in the path that performs the time-delay attack. After receiving the packet P_n forwarded by N_2, N_3 maliciously introduces a delay before forwarding P_n to the next node N_4. This malicious behavior eventually causes N_{Sink} to receive the outdated packet P_n, which brings significant damage to the real-time scenarios in IoT networks. During the transmission of packets, it is difficult to determine the presence of malicious nodes because of the multiple nodes involved in forwarding and the possibility of inherent communication delays. What is worse, if a relay node close to the Sink is malicious, it can affect the normal arrival of massive packets from multiple upstream nodes.

Meanwhile, Time-delay attacks are challenging to detect. Most other attacks in the IoT environment violate the communication protocol to some extent or tamper with the original content of the packets [8]. However, a malicious node performing time-delay attacks only delays the forwarding of packets. It does not require any decryption or tampering of the packet content, nor does it violate the established protocols of the network. Thus, it cannot be detected using traditional cryptographic schemes [14]. Moreover, the attacker does not need much prior knowledge of the system. The attack can be easily implemented at the network layer with low costs.

To address the above issues, the naive idea is to collect the processing time of packets for each node and the system features that affect the processing time of packets, such as buffer queue length at the time of sending, collision count, NOACK(no ACK is received) count, Received Signal Strength Indicator (RSSI), and Link Quality Indicator (LQI). Then, time-delay attacks can be detected by modeling the relationship between system features and processing time. However, it is difficult to formalize the relationship using a general mathematical formula since various system features can affect the packet's processing time. Machine learning methods can help to solve this problem by autonomously modeling the relationship between multiple variables [15]. Therefore, we first propose a baseline algorithm that uses a machine learning model to predict the processing time of packets. By detecting each node using the baseline algorithm, it is possible to determine whether the time-delay attack exists in the IoT network.

However, there are still two shortcomings with the baseline algorithm.

1. Since the above detection process collects data from all nodes, which can result in a large amount of energy consumption. In fact, not all data are useful for detection, and it is not efficient to collect the data from all nodes without selectivity.
2. Time-delay attacks are flexible and changeable with different attack patterns. The machine learning models with the best detection performance under different attack patterns are often different, it is not robust enough to use a single model.

Contributions To alleviate the two shortcomings mentioned above, we propose a Detection algorithm based on Node pruning and Model fusion (DNM). Firstly, DNM

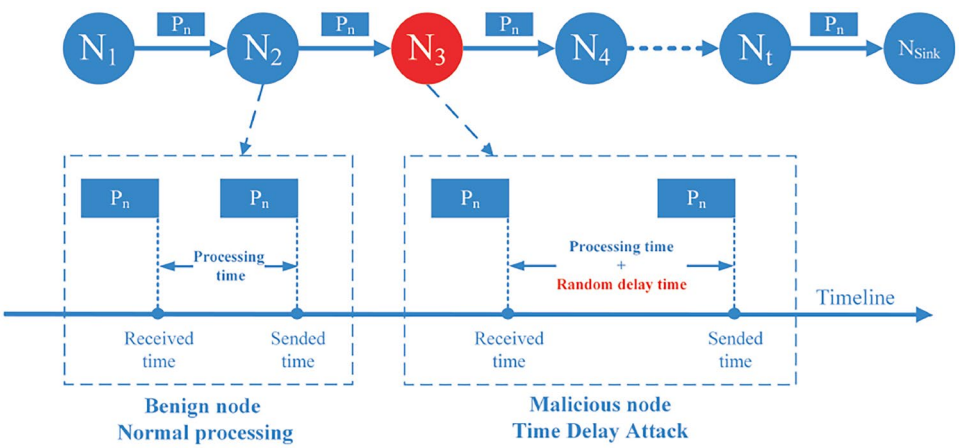

Fig. 1 Differences in behavior between benign and malicious nodes when forwarding packets

significantly reduces energy consumption by the node pruning mechanism. The Sink first groups the received packets according to their routing paths, and an outlier detection algorithm is used to obtain the reputation value of each path group. Then, combined with the diversity of paths, the global reputation value of each node can be further calculated, based on which the suspected nodes can be filtered out. The Sink only needs to collect the data recorded by the suspected nodes, thereby reducing the number of transmitted packets and achieving energy savings. Secondly, the model fusion mechanism in DNM can achieve the best detection performance. The Sink fuses several models that perform better in the historical data into a new fusion model. The fusion model is then used for malicious node detection. The model fusion mechanism can dynamically select better models for detection in the case of increasing data volume of historical data, thus showing the best detection performance. It is worth noting that the training and detection of the model are performed on the Sink (the resource-rich node) and do not impose additional overhead on other nodes.

In summary, the contributions of this paper are as follows:

- A stealthy attack called the time-delay attack is introduced. Meanwhile, a baseline detection algorithm based on machine learning is proposed.
- We design a Detection algorithm based on Node pruning and Model fusion (DNM) based on the baseline algorithm. DNM significantly reduces energy consumption through the node pruning and improves the detection performance by the model fusion.
- The experimental results show that DNM has a lower overhead and higher detection performance than the baseline algorithm.

Organization The rest of the paper is organized as follows. Section 2 presents malicious node detection in IoT networks and time-delay attacks detection in other fields. Section 3 models the system, and Section 4 presents the baseline algorithm. Section 5 describes the DNM detection algorithm proposed in this paper, and Section 6 gives the experimental results. Finally, the conclusion is conducted in Section 7.

2 Related work

Nowadays, it is still a challenge to detect internal attacks in IoT networks [16], so we first present related studies on internal attacks. To the best of our knowledge, there is no research on time-delay attacks in IoT networks. We then discuss related research in other networks. In addition, the behavior of node is usually evaluated to detect malicious nodes in IoT networks, we finally introduce the trust-based detection.

2.1 Internal attacks in IoT

The IoT network routing faces many security challenges, such as selective forwarding attacks, flooding attacks, wormhole attacks, black hole attacks, and Sybil attacks.

Huang et al. [17] developed an artificial immune system based on a danger model to detect selective forwarding attacks. Their scheme first obtains danger signals from multiple dimensions and then uses support vector machines to filter out suspected selective forwarding attacks from denial-of-service attacks. In [18], Ding et al. used a reinforcement learning algorithm to model selective forwarding attacks by malicious nodes. To improve the robustness of the detection method, they designed a double-threshold density peak clustering algorithm. Since malicious nodes cause persistent anomalies, suspicious nodes that are anomalous can be identified from normal nodes. Chen et al. [19] identified flooding attacks by modeling the interaction between the two as a two-person Bayesian game to model the behavior of attackers and defenders accurately. Then, the attacker's rational behavior and the defender's optimal strategy are revealed by deriving Bayesian Nash equilibrium points. Inspired by the obtained Bayesian Nash equilibrium, they proposed a cost-effective defense decision framework. In order to detect HELLO flooding attacks, a new robust model for using optimized deep learning methods was proposed in [20]. Cluster head selection, k-path generation, HELLO flooding attack detection and prevention, and optimal shortest path selection are the steps used in this model.

Teng et al. [21] proposed a wormhole detection algorithm combined with a node trust optimization model to detect wormhole attacks. The method first adds the nodes whose neighbors exceed the threshold to the list of suspicious nodes, and then the exclusive neighbors of suspicious nodes communicate with each other. The paths with hop counts exceeding the wormhole threshold are marked as paths to be tested. In [22], a detection mechanism based on lightweight Bloom filters and physically unclonable functions was proposed to detect Sybil attacks. This approach aims to minimize memory cost and detection latency without affecting detection accuracy. Alghamdi et al. [23] proposed a wormhole detection method based on joint deep-learning techniques and dynamic trust factors. Their detection method is based on two trust attributes. Convolutional neural networks and long and short term memory deep learning models have been trained using a federated approach to ensure data security and privacy at the node level. Kim et al. [24] proposed a trust path routing scheme based on physical identification to detect Sybil attacks. This scheme uses the received signal strength indicator and a centralized trust scheme to improve Sybil node detection.

2.2 Time-delay attack

The current research on time-delay attacks mainly focuses on the field of time synchronization, Cyber-Physical Systems (CPS), and Networked Control Systems (NCS). In [14], for time-delay attacks on the time synchronization of wireless sensor networks, Song et al. used the generalized extreme learning deviation (GESD) algorithm and detection by time conversion techniques. Moradi et al. [25] proposed a corresponding solution to the delay attack on synchronization messages in the Precision Time Protocol (PTP), a time synchronization protocol. Since PTP is vulnerable to network attacks against its components and synchronization services, Moussa et al. [26] proposed a protocol to detect delay attacks against PTP. The nodes calculate the offset using the information in the received Report messages. They argue that the calculated offset should match a pre-specified threshold, and results above the threshold indicate that the node may suffer from delay attacks. In the field of time synchronization, time synchronization between nodes mainly relies on the transmission of specific message packets between a host and its neighboring slaves by some protocol. It does not involve the multi-hop transmission of packets in the IoT environment.

Both studies [27, 28] demonstrated that time-delay attacks can seriously impact cyber-physical systems like power grids. And both studies [29, 30] proposed corresponding detection methods for time-delay attacks under cyber-physical and networked control systems. Ganesh et al. [31] proposed a deep learning-based approach to detect time-delay attacks. They designed a hierarchical Long and Short-Term Memory (LSTM) model to process the raw data stream from the associated CPS sensors and continuously monitored the embedded signals in the data to detect the attacks. In Cyber-Physical Systems and Networked Control Systems, mostly message packets are transmitted from the controller to the actuator or from the sensors to the controller, without involving multi-hop transmission of packets as in IoT. Moreover, most studies assumed that the delay time is fixed or linearly varying [25, 28, 32, 33]. These papers do not consider that a powerful attacker can disguise the attack as a normal communication delay by setting a random delay time. In summary, due to the multi-hop nature of IoT networks, the above research methods for time-delay attacks in other fields cannot be applied to IoT.

2.3 Detection based on trust value mechanism

The node pruning in DNM is based on the trust value mechanism, and many literatures have demonstrated its effectiveness in IoT malicious node detection. To identify malicious nodes that cause attacks in smart city applications and networks, Altaf et al. [34] proposed a trust evaluation system model for detecting On-Off attacks. The total trust value of nodes is obtained by collecting direct observations of communicating nodes and suggestions from neighboring nodes. In addition, the contextual similarity metric is calculated to filter out those nodes that constitute Sybil attacks. A multi-level trust intelligence scheme based on cryptographic authentication was proposed in [35]. In this scheme, each node in the network uses RREQ packets to obtain the trust value of each node by evaluating the behavior of neighboring nodes. The control packets are then used to discover and eliminate malicious gray hole nodes. In [10], the authors used random forest and subjective logic theory to construct a trust model in IoT networks. The model is used to address Sinkhole attacks based on low-power and lossy networks. They proposed RPL routing protocol, RFTrust, is invoked to circumvent malicious nodes only when the trust level of neighboring nodes becomes low, thus reducing the extra overhead and energy consumption.

In [36], Liu et al. used perceptron algorithm and K-means method to calculate the trust value of nodes in the network for identifying three typical attacks in IoT: tampering attacks, drop attacks, and replay attacks. The method first classifies nodes into benign, unknown, and malicious groups based on trust values. For the unknown group, further detection is performed by optimizing the path. This idea is similar to the DNM detection algorithm proposed in this paper. After first selecting the suspected nodes by node pruning mechanism at a small cost, further detection of the suspected nodes is carried out. To identify conditional packets manipulation attack in IoT networks [37], the malicious node detection framework CPMAED was proposed. This framework uses regression and clustering algorithms to evaluate the trust value of each relay node and classify them as benign and malicious. For multiple-mix-attack in IoT networks, Ma et al. [38] proposed a method called distributed consensus-based trust model (DCONST) to identify multiple-mix-attack. The method allows assessing the trustworthiness of IoT nodes by sharing specific information called cognition. The node trustworthiness values are clustered by the K-Means clustering method to detect malicious nodes and analyze their specific attack behavior. In large-scale clustered wireless sensor networks, Singh et al. [39] proposed a lightweight trust mechanism to protect such systems from various malicious attacks. A dynamic trust update algorithm based on parameter trust priority is also proposed to reward or penalize the trust value of nodes.

3 System model

In this section, we formalize the network model, and model the time-delay attack. In addition, Table 1 shows a list of notations for reference.

3.1 Network Model

A typical IoT network consists of many nodes (e.g., sensors). The nodes exchange information with each other through wireless communication. Due to the limited communication range of a single node, packet transmission between different nodes usually relies on neighboring nodes for forwarding.

We denote N_{set} as the set of nodes. N_{set} consists of Sink nodes that collect sensor data and sensor nodes with different functions, which can be represented as:

$$N_{set} = \{N_{Sink}, N_{sensor}\} \tag{1}$$

The set of Sink nodes N_{Sink} consists of m resource-rich devices, which can be represented as:

$$N_{Sink} = \{N_{Sink_1}, N_{Sink_2}, \ldots, N_{Sink_m}\} \tag{2}$$

Without loss of generality, in this paper, we only study the case where there is one Sink node in the network, which is similar to other works [10, 35–39]. It is worth noting that our research approach can be extended to IoT networks with multiple Sink nodes.

Table 1 Notations

Symbol	Meaning
N_{set}	The set of nodes
N_{Sink}	Sink nodes in N_{set}
N_{sensor}	Sensor nodes in N_{set}
N_i	A node of N_{sensor}
Pkt_{N_i}	Packets sent from N_i
Pkt_j	A packet in Pkt_{N_i}
$Path_{N_i}$	The set of paths through which N_i sends packets
$Path^q_{N_i}$	The q-th path in $Path_{N_i}$
$Path^q_{N_i}[k]$	The k-th node in $Path^q_{N_i}$
N_{ma}	The set of hijacked malicious nodes in N_{sensor}
N^m_{ma}	The m-th malicious node in N_{ma}
ε^m_k	The probability that N^m_{ma} performs time-delay attack
α^m_k	Delay time for N^m_{ma} to perform time-delay attack
β_k	The normal communication delay
τ	Maximal delay time for malicious delay
T^{ave}_{delay}	The average delay time
R^{tab}_{send}	Sending packet information record table
P_{id}	The unique identification number of a packet
T_{send}	Time of successful transmission of a packet
N_{next}	The target next hop of a packet in a network route
$N_{previous}$	The previous hop of a packet in a network route
$R^{tab}_{forward}$	Forwarding packet information record table
$T_{receive}$	Time of successfully receive a packet
V_n	A characteristic value
$R^{tab}_{receive}$	Receiving packet information record table
S_{id}	The unique identification number of the source node
$T^{node_i}_{Sink}$	The total duration of a packet

The set of sensor nodes N_{sensor} consists of n resource-constrained devices, which can be represented as:

$$N_{sensor} = \{N_1, N_2, \ldots, N_i, \ldots, N_n\} \tag{3}$$

where N_i represents a node in N_{sensor}. Each node in the set N_{sensor} initially has equal energy and other resources, but the detection tasks may be different. Sensor nodes periodically send packets to the Sink node for data collection.

The packets with N_i as the source node and N_{Sink} as the destination node can be represented as:

$$Pkt_{N_i} = \{Pkt_1, Pkt_2, \ldots, Pkt_j, \ldots, Pkt_n\} \tag{4}$$

where Pkt_j represents a packet in Pkt_{N_i} sent from N_i.

The packet Pkt_j has multiple paths from the source node N_i to the destination node N_{Sink}, which can be expressed as:

$$Path_{N_i} = \{Path^1_{N_i}, Path^2_{N_i}, \ldots, Path^q_{N_i}, \ldots, Path^m_{N_i}\} \tag{5}$$

where $Path^q_{N_i}$ is denoted as the q-th path from N_i to N_{Sink}, which can be expressed as:

$$Path^q_{N_i} =< N_i, \ldots, N_j, \ldots, N_k, \ldots, N_{Sink} > \tag{6}$$

where N_j and N_k represent the nodes in N_{sensor}. Taking Fig. 2 as an example, from the node N_2 to the node N_{Sink}, there are two paths: $Path^1_{N_2} =< N_2, N_6, N_{Sink} >$ and $Path^2_{N_2} =< N_2, N_3, N_5, N_{Sink} >$. $Path^1_{N_2}$ means the packets from N_2 follow the path: $N_2 \rightarrow N_6 \rightarrow N_{Sink}$, and $Path^2_{N_2}$ means N_2 transmits the packets along the path: $N_2 \rightarrow N_3 \rightarrow N_5 \rightarrow N_{Sink}$.

Meanwhile, $Path^q_{N_i}[k]$ denotes the k-th node in $Path^q_{N_i}$. For example, $Path^1_{N_2}[1] = N_2$.

3.2 Attack model

In our attack model, the attacker is assumed to have the following characteristics:

- The attacker is powerful enough to capture and manipulate one or more legitimate nodes remotely [10, 35–39].

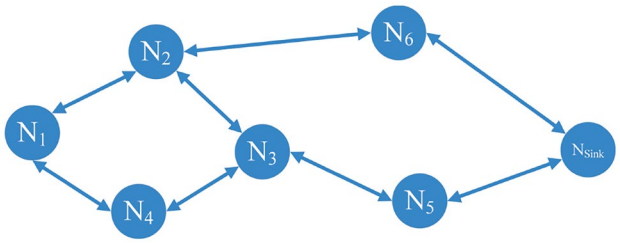

Fig. 2 A typical IoT network structure

An attacker can use captured nodes to launch time-delay attacks.

- Malicious nodes only delay packet forwarding within a certain time range. If the delay time is too long, the time-delay attack is equivalent to the drop attack, which can be easily detected by the existing drop attack [40].
- The attacker's goal is to affect the entire system through a few malicious nodes in a highly stealthy manner. The attacker does not interfere with normal network operations, such as changing data records, altering packets, compromising network devices, and tampering with key management operations. Intrusion Detection System (IDS) can detect such activities, which exposes the attacker to the risk of being detected [41].

Next, we provide a formal description of the attack model. In an IoT network, there is a path $Path^q_{N_i}$ from node N_i to node N_{Sink}, and the number of relay nodes in $Path^q_{N_i}$ is y. We assume that there are x ($1 \leqslant x \leqslant y$) malicious nodes in $Path^q_{N_i}$, denoted as:

$$N_{ma} = \{N^1_{ma}, N^2_{ma}, \ldots, N^m_{ma}, \ldots, N^x_{ma}\} \tag{7}$$

where N^m_{ma} is the m-th malicious node in N_{ma}.

Malicious nodes in N_{ma} perform time-delay attacks on packets $Pkt_{N_i} = \{Pkt_1, Pkt_2, \ldots, Pkt_k, \ldots, Pkt_n\}$ from source node N_i to destination node N_{Sink}. N_{ma} performs the time-delay attack on the k-th packet Pkt_k ($1 \leqslant k \leqslant n$) with probability $\{\varepsilon^m_1, \varepsilon^m_2, \ldots, \varepsilon^m_k, \ldots, \varepsilon^m_n\}$ ($0 < \varepsilon^m_k < 1$), and ε^m_k conforms to a random distribution. The delay time of N_{ma} for Pkt_k is $\{\alpha^m_1, \alpha^m_2, \ldots, \alpha^m_k, \ldots, \alpha^m_n\}$ ($0 < \alpha^m_k < \tau$), where τ is the maximum delay time, and α^m_k conforms to a random distribution within its allowed range. If the normal communication delay from N_i to N_{Sink} is $\{\beta_1, \beta_2, \ldots, \beta_k, \ldots, \beta_n\}$, the average delay time in the presence of time-delay attacks is denoted as:

$$T^{ave}_{delay} = \frac{1}{n} \sum\nolimits_{i=1}^{n} \left(\sum\nolimits_{m=1}^{x} (\varepsilon^m_i \times \alpha^m_i) + \beta_i \right) \tag{8}$$

Note that even in the absence of an attacker, packets may be delayed to reach the destination node due to normal communication delays. The attacker delays the packets with a certain probability ε and random delay duration α to disguise the attack as normal communication delays, which is more stealthy and hard to be detected.

4 Baseline detection algorithm for time-delay attacks

In this section, we first introduce the basic framework of the baseline algorithm. Then the primary process of the baseline algorithm is described, including: (1) the collection and processing of packet delivery and node context information. (2) forwarding delay model training and malicious node detection.

4.1 The framework of baseline algorithm

In an IoT network, sensor nodes periodically report the latest sensor data to the Sink. Due to the limited communication range of sensor nodes, the packets require multiple relay nodes for forwarding. In order to detect time-delay attacks, the relay nodes' forwarding behavior needs to be monitored. As shown in Fig. 3, the baseline algorithm consists of two main stages:

4.1.1 The collection and processing of packet delivery and node context information (see Section 4.2 for details)

To monitor the forwarding behavior of nodes, each node records packet delivery and node context information when processing packets. Specifically, packet delivery information is the time when nodes send, forward, or receive packets. Node context information is the context when nodes forward packets, such as RSSI, LQI, and the number of packets in the buffer queue. Each node records the above information locally and sends them to the Sink periodically. After the information arrives at the Sink, the Sink fuses the packet delivery information into the forwarding delay. And then, node context information and forwarding delay are combined as the training set for model training.

As shown in Fig. 3, it assumes that node N_1 sends the packet Pkt_i to the Sink N_{Sink} through node N_3, where the time when N_1 sends Pkt_i (t_1), the time when N_3 receives and sends Pkt_i (t_2 and t_3), and the time when N_{Sink} receives Pkt_i (t_4) are the packet delivery information. After the Sink receives these information, it can calculate the forwarding delay of N_3 forwarding Pkt_i, which equals $t_4 - t_1$. Additionally, the information such as RSSI, LQI, and the number of packets in the buffer when N_3 forwards Pkt_i is the node context information.

4.1.2 Forwarding delay model training and malicious node detection (see Section 4.3 for details)

We assume that the IoT network does not suffer from malicious attacks in some stages [42], e.g., the initialization stage of devices. Therefore, we collect and process the information from these stages into training sets for training the forwarding delay models. Then, the Sink collects and processes such information from other stages into detection sets for malicious node detection using the trained models. We build different models for different relay paths of each relay node. As shown in Fig. 3, there are two relay paths for relay node N_3: $N_1 \rightarrow N_3 \rightarrow N_{Sink}$ and $N_2 \rightarrow N_3 \rightarrow N_{Sink}$. Each relay path

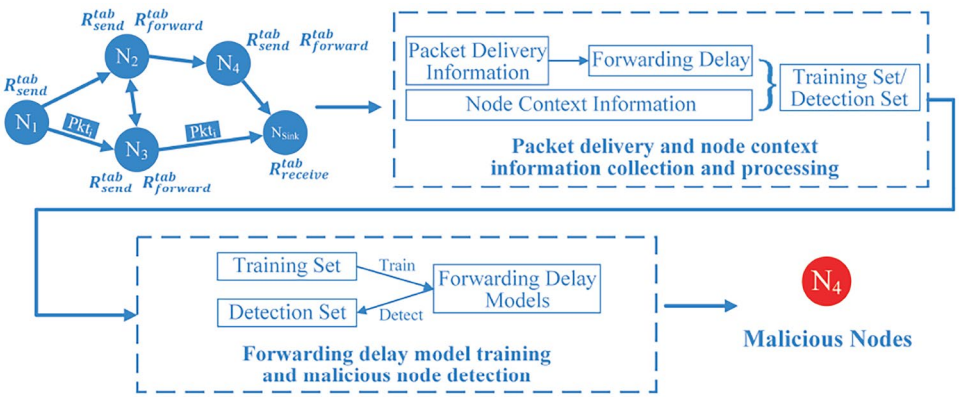

Fig. 3 The flow of baseline algorithm

corresponds to a forwarding delay model. Finally, multiple forwarding delay models of the relay node vote on whether the node is malicious or not.

4.2 The collection and processing of packet delivery and node context information

The packet routing process involves multiple nodes, and there may be several malicious nodes among these nodes. To accurately find all malicious nodes, the behavior of each relay node needs to be detected. Therefore, we collect and process the information when the packets pass through each relay node. As shown in Fig. 4, the collection and processing of packet delivery and node context information consist of four main stages:

1. Collecting packet delivery and node context information.
2. Getting the routing path of each packet.
3. Extracting the samples based on routing paths.
4. Aggregating the samples of relay nodes as datasets.

4.2.1 Collecting packet delivery and node context information

To monitor packet forwarding among the nodes, each node maintains the following three tables when sending, forwarding, and receiving packets:

- Sending Packet Table (R^{tab}_{send}): Each node in N_{sensor} stores R^{tab}_{send}, which contains three fields: $< P_{id}, T_{send}, N_{next} >$, where P_{id} represents the unique sequence number of packets transmitted in the network, T_{send} records the time of sending packets, and N_{next} represents the next-hop node of packets. Whenever a node sends a packet, a record is added to the local R^{tab}_{send}.
- Forwarding Packet Table ($R^{tab}_{forward}$): All non-leaf nodes in N_{sensor} store $R^{tab}_{forward}$, which contains the following fields: $< P_{id}, T_{receive}, T_{send}, N_{next}, V_1, V_2, \cdots, V_n >$. The

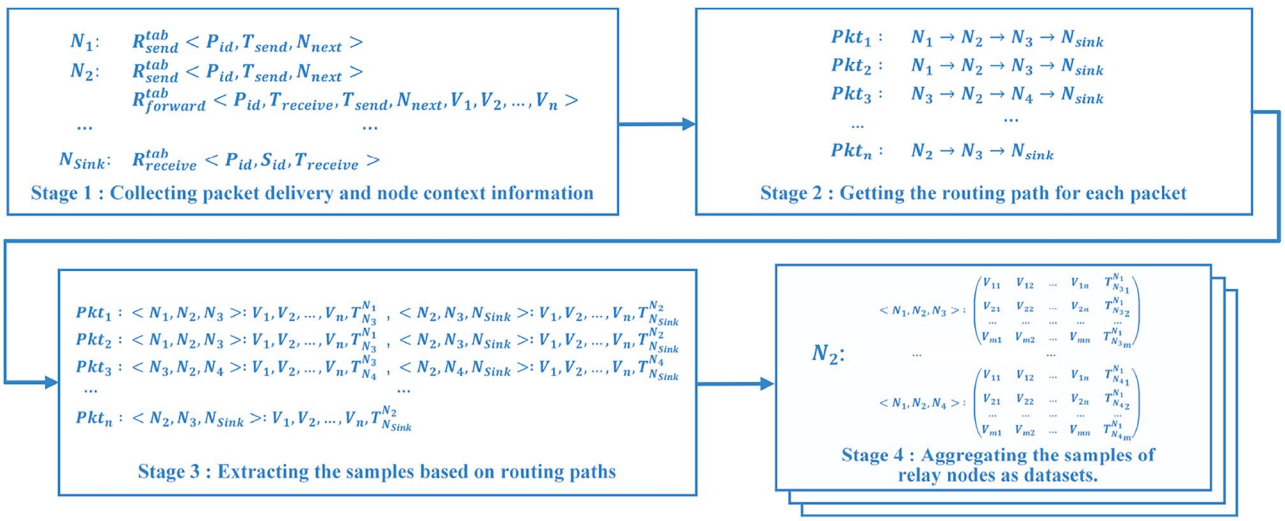

Fig. 4 Packet delivery and node context information collection and processing

meanings of P_{id}, T_{send}, and N_{next} are the same as described in R^{tab}_{send}. Additionally, $T_{receive}$ records the time of receiving packets, and $V_1, V_2, \ldots, V_n$ represents the context at the time of the packet forwarding, such as RSSI, LQI, the number of packets in the N_{sensor}'s buffer queue, the times of packet collision, the times of NOACK during packet sending, and the total time of NOACK during packet sending. Every time a node forwards a packet to its next-hop node, the above information is added to the local $R^{tab}_{forward}$.

- Receiving Packet Table ($R^{tab}_{receive}$): Every time the Sink successfully receives a packet, it stores three fields: $< P_{id}, S_{id}, T_{receive} >$, where S_{id} represents the identification of the source node obtained from the protocol stack. The meanings of P_{id} and $T_{receive}$ are the same as described in $R^{tab}_{forward}$.

Due to the limited storage resources of sensor nodes, the records in the above tables are overwritten on a first-in-first-out basis. Meanwhile, R^{tab}_{send} and $R^{tab}_{forward}$ stored by each node are sent to the Sink at a regular interval.

4.2.2 Getting the routing path of each packet

After the Sink receives R^{tab}_{send}, $R^{tab}_{forward}$, and $R^{tab}_{receive}$, we can get the routing path of each packet by looking up the above tables. Specifically, for a packet Pkt_i sending from the source node N_s to the Sink: we first take out a record $< P_{id}, T_{send}, N_{next} >$ from N_s's R^{tab}_{send}, and according to N_{next} of this record, we can find the next-hop node of the route, denoted as N_n. In N_n's $R^{tab}_{forward}$, we find the record where $P_{id} = Pkt_i$'s P_{id}, and take out N_{next} in that record to get the next-hop node of the route. Repeat this process until the next-hop node is the Sink, and finally, we can get a complete routing path of Pkt_i.

For example, as shown in Fig. 3, the packet Pkt_i is sent from N_1 to N_{Sink} through N_3. After the Sink collects the tables of all nodes, there exists a record $< P_{id}, T_{send}, N_3 >$ in N_1's R^{tab}_{send}, and we can know the next-hop node of the route is N_3. Then we find out the record $< P_{id}, T_{receive}, T_{send}, N_{Sink}, V_1, V_2, \cdots, V_n >$ from N_3's $R^{tab}_{forward}$, and we can know the next-hop node of the route is N_{Sink}. Finally, the routing path of Pkt_i can be obtained as: $N_1 \rightarrow N_3 \rightarrow N_{Sink}$.

4.2.3 Extracting the samples based on routing paths

The training samples of forwarding delay models comprise node context information (as features) and forwarding delay (as a label). The node context information is recorded by the relay nodes themselves, while the forwarding delay is derived by fusing packet delivery information. How to select the appropriate packet delivery information for fusion to derive the forwarding delay is an important issue. The malicious nodes may tamper with the receiving or sending time of the packets ($T_{receive}$ and T_{send} in $R^{tab}_{forward}$) recorded by themselves. It means the packet delivery information recorded by the relay nodes themselves cannot be trusted. Therefore, we use the packet delivery information recorded by two neighboring nodes of the relay node to derive the relay node's forwarding delay.

As shown in Fig. 5, packet Pkt_i is sent from N_1 to N_{Sink} through N_j, N_i, and N_k. When N_j forwards Pkt_i, a record $< P_{id}, T^j_{receive}, T^j_{send}, N_i, V^j_1, V^j_2, \cdots, V^j_n >$ is added to N_j's $R^{tab}_{forward}$. When N_k forwards Pkt_i, a record $< P_{id}, T^k_{receive}, T^k_{send}, N_l, V^k_1, V^k_2, \cdots, V^k_n >$ is added to N_k's $R^{tab}_{forward}$. The forwarding delay $T^{N_j}_{N_k}$ of N_i forwarding Pkt_i is calculated from T^j_{send} and $T^k_{receive}$:

$$T^{N_j}_{N_k} = T^k_{receive} - T^j_{send} \tag{9}$$

There are multiple relay nodes on the packet's routing path, and we extract samples for each relay node separately. For example, as shown in Stage 3 of Fig. 4, there are two relay nodes in Pkt_1's routing path: N_2 and N_3. For a relay path $< N_1, N_2, N_3 >$, we can extract N_2's node context information as a sample. For another relay path $< N_2, N_3, N_{Sink} >$, we can extract N_3's node context information as a sample. These two samples are used to train the forwarding delay models for N_2 and N_3, respectively.

4.2.4 Aggregating the samples of relay nodes as datasets

Since time synchronization protocols are vulnerable to time-delay attacks [14, 43], this paper considers the more general case that the nodes are not synchronized in time. Therefore, we build different forwarding delay models for relay paths and detect them separately. To match each forwarding delay model with its training data, we aggregate the samples with the same relay path into a dataset for model training. For example, as shown in Stage 4 of Fig. 4: for the relay node N_2, we aggregate the samples of two relay path ($< N_1, N_2, N_3 >$ and $< N_1, N_2, N_4 >$) into two datasets, which are used for the training of two different forwarding delay models.

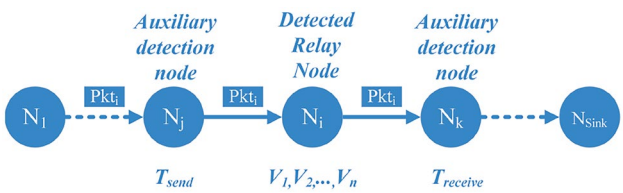

Fig. 5 The way to obtain the forwarding delay of the relay node

4.3 Forwarding delay model training and malicious node detection

After obtaining the datasets of relay paths, we use them to train forwarding delay models. The forwarding delay model is trained using classical machine learning algorithms such as Support Vector Regression (SVR) and Decision Tree (DT). Then, multiple forwarding delay models of the relay node vote to decide whether the node is malicious or not. As shown in Fig. 6, the training and detection of the forwarding delay model can be divided into three main stages:

4.3.1 Training process

A relay node has multiple relay paths, and we build the forwarding delay model for each relay path. We randomly divide the dataset of relay paths into a training set and a test set. The training set is used to train the forwarding delay model, and the trained model is tested by the test set. Finally, the loss of the model can be obtained, which is denoted as $loss_{training}$. As shown in Stage 1 of Fig. 6, for the relay node N_i, the forwarding delay models are trained separately for N_i's relay paths: $< N_j, N_i, N_k >$, $< N_j, N_i, N_l >$ and $< N_j, N_i, N_m >$, and the loss of each model is obtained.

4.3.2 Detection process

Similar to the training process, the Sink collects packet delivery and node context information from the IoT network, which is processed to obtain multiple detection datasets. Each detection dataset is handed over to its corresponding forwarding delay model for prediction. Moreover, the loss can be calculated based on the predicted and actual values, denoted as $loss_{detection}$.

4.3.3 Voting for malicious nodes

For each forwarding delay model of N_i, $loss_{training}$ and $loss_{detection}$ obtained in the training and detection process are compared, and their difference is used to vote whether N_i is malicious. The final decision that whether N_i is malicious or not depends on the voting result of N_i's multiple forwarding delay models.

5 Detection algorithm based on Node pruning and Model fusion for time-delay attacks

The baseline algorithm collects packet delivery and node context information from all nodes to detect malicious nodes, which causes unnecessary energy consumption. In fact, not all nodes' information is required to be collected and detected. Meanwhile, the time-delay attack has various attack patterns. The single machine learning model used in the baseline algorithm cannot cope well with the different attack patterns, resulting in poor detection performance. To reduce the energy consumption and improve the detection performance of the baseline algorithm, we designed an efficient Detection

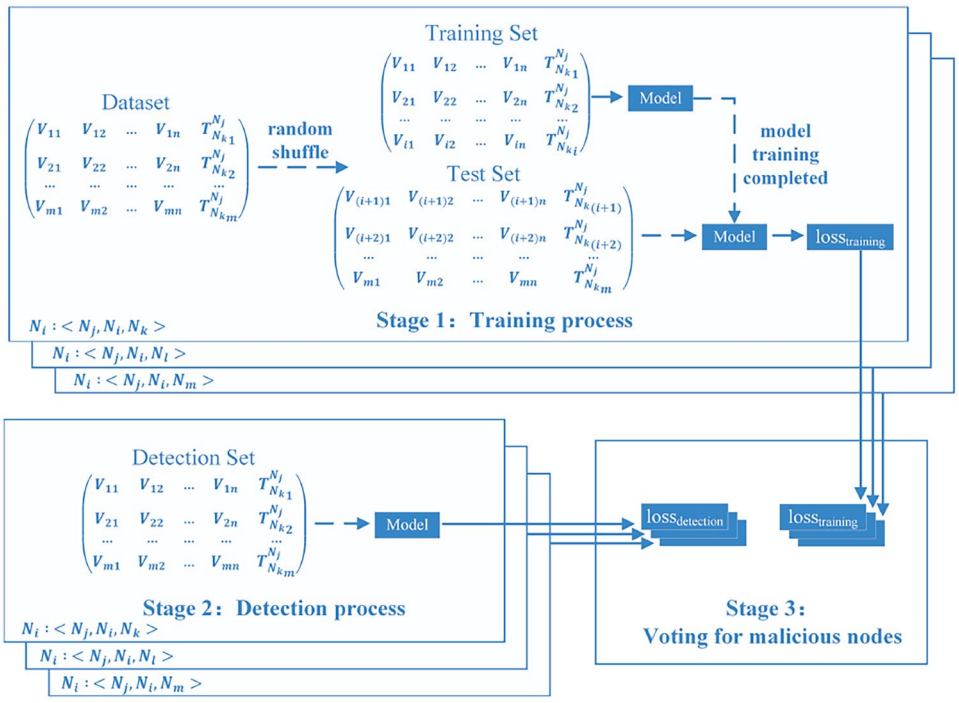

Fig. 6 Forwarding delay model training and detection process

algorithm based on Node pruning and Model fusion (DNM). In this section, we first describe the overall architecture of DNM, and then give a detailed description.

5.1 The overall architecture of DNM

The overall architecture of DNM is shown in Fig. 7, which consists of two main parts: node pruning and model fusion.

5.1.1 Node pruning

The baseline algorithm collects information from all nodes without selectivity, resulting in unnecessary energy consumption. In DNM, we do not directly collect the information of all nodes, but first filter out some suspected nodes through node pruning instead. Node pruning only uses the routing information of packets to filter out suspected nodes with a small overhead. First, the Sink groups the received packets by their routing paths, and the packets with the same routing path are grouped into the same path group. Then, for each path group, an outlier detection algorithm is performed on the transmission time of each packet. The outlier frequency is used as the basis for assigning a reputation value to each relay node in the path group. Finally, by combining each relay node's reputation value in different path groups, a global reputation value of the relay node can be obtained. The nodes with low global reputation values are considered as suspected nodes.

5.1.2 Model fusion

An attacker may execute the time-delay attack with different attack probabilities and strengths. A single machine learning model used in the baseline algorithm cannot handle complex attack scenarios well, resulting in non-robust results. In DNM, we improve the detection performance by model fusion. We pick several machine learning models that perform well in test sets and combine them into a fusion model. For the suspected nodes filtered by node pruning, the Sink selectively collects packet delivery and node context information from the suspected nodes and processes them into datasets. The datasets are then fed into the fusion model to determine whether the suspected nodes are malicious.

5.2 Node pruning

The process of node pruning is shown in Fig. 8, which consists of the following four main stages:

1. Getting the routing path of all packets through lightweight packet path tracing algorithm.
2. Grouping the packets according to their routing paths.
3. Assigning reputation values to routing paths based on the outlier detection results.
4. Aggregating reputation values of the relay nodes.

5.2.1 Getting the routing path of all packets through lightweight packet path tracing algorithm

The baseline algorithm uses N_{next} in $R^{tab}_{forward}$ to implement packet path tracking (see Section 4.2.2 for details), which relies on collecting node context information from all nodes and has a significant overhead. To enable packet path tracking at a minor cost, we introduced Provenance-enabled Packed Path Tracing (PPPT) approach [44] in DNM. PPPT is a lightweight packet path tracing scheme. Packet path tracking is achieved by querying the relevant information stored in each node. Its average power consumption and memory overhead are almost negligible [44].

PPPT can obtain the complete path of packets in the network by combining node-level provenance and system-level provenance. The node-level provenance is introduced by embedding a previous-hop node's identification and a unique sequence number of packets in the corresponding relay node. In DNM, we reuse the previous-hop node's identification and the unique sequence number of packets in $R^{tab}_{forward}$, denoted as $N_{previous}$ and P_{id}. Then $R^{tab}_{forward}$ has the following fields: $< P_{id}, T_{receive}, T_{send}, N_{previous}, N_{next}, V_1, V_2, \cdots, V_n >$. In

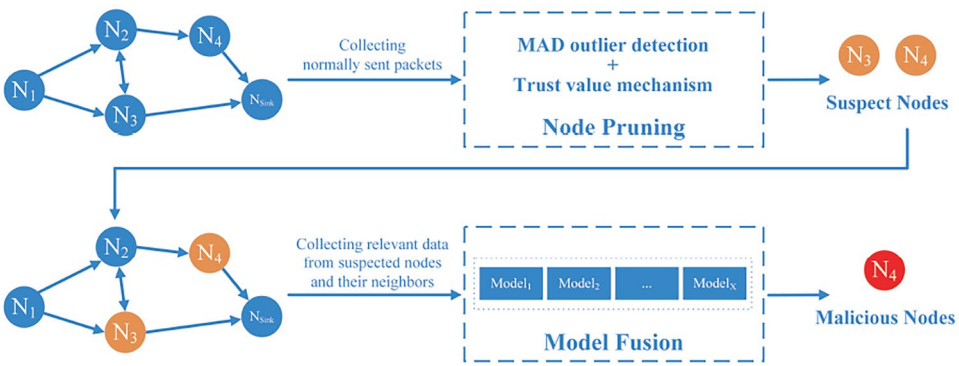

Fig. 7 DNM algorithm flow

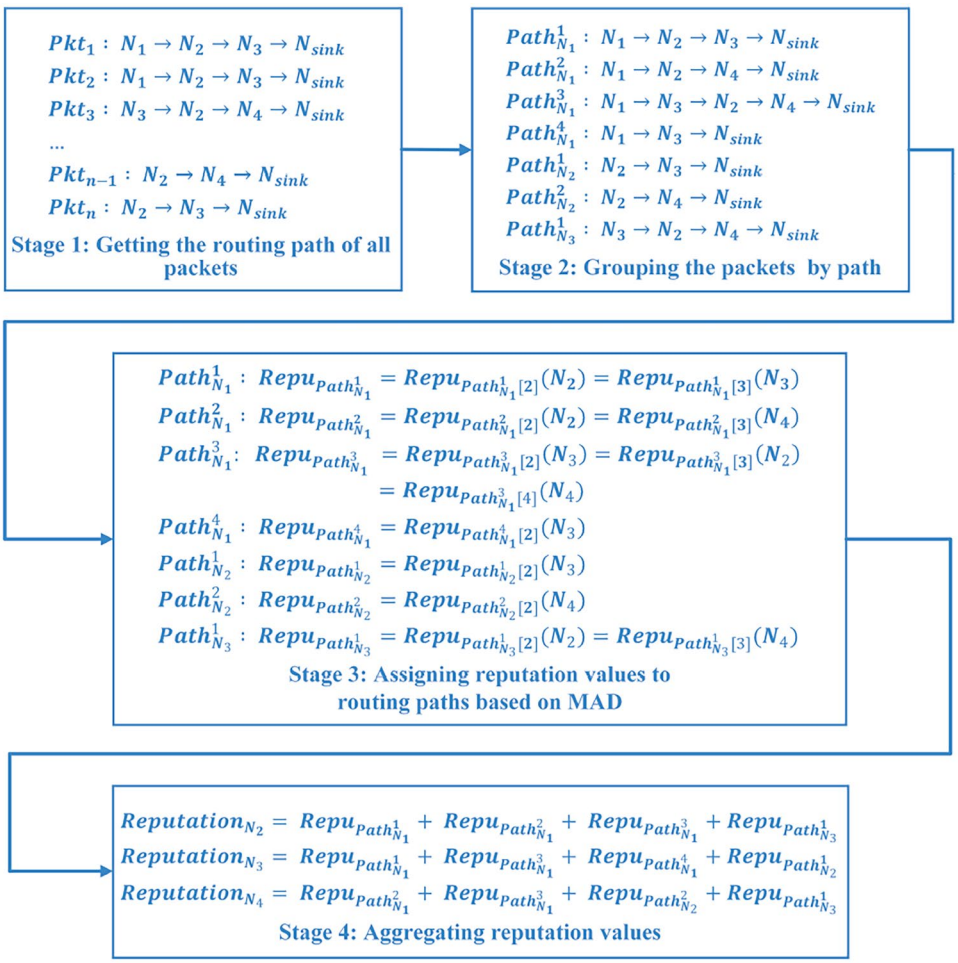

Fig. 8 Node pruning

addition to the node-level provenance, a system-level provenance is included to capture the complete packet trace by including destination and source node identification pairs.

5.2.2 Grouping the packets according to their routing paths

After getting the routing path of each packet, we group the packets according to their routing paths. As shown in Stage 2 of Fig. 8, there are two paths from N_2 to N_{Sink}: $N_2 \rightarrow N_3 \rightarrow N_{Sink}$ ($Path^1_{N_2}$) and $N_2 \rightarrow N_4 \rightarrow N_{Sink}$ ($Path^2_{N_2}$). Therefore, the packets sent from N_2 to N_{Sink} are divided into two different groups.

5.2.3 Assigning reputation values to routing paths based on the outlier detection results

For the transmission time of packets on the same routing path, the frequency of outliers remains stable even in the presence of normal communication delays. However, malicious nodes in the path may delay the packet forwarding, which can result in more packet transmission time outliers in the path. If the outlier frequency of the packet transmission time is somewhat higher than the historical outlier frequency, the path is considered as a suspected path. Based on the outlier detection results, a global reputation value is assigned to relay nodes in the routing path.

Different from the baseline algorithm, R^{tab}_{send} is no longer used in DNM to record the information of packet sending time T_{send}, instead T_{send} is carried in the packet sent by each node. The Sink still uses $R^{tab}_{receive}$ to record the receiving packet information and $T_{receive}$ in $R^{tab}_{receive}$ to record the packet receiving time. For a packet Pkt sent to the Sink from N_i, the packet transmission time $T^{N_i}_{Sink}$ can be obtained from $T^{Sink}_{receive}$ (the time when the Sink receives Pkt) and T^{i}_{send} (the time when N_i sends Pkt):

$$T^{N_i}_{Sink} = T^{Sink}_{receive} - T^{N_i}_{send} \tag{10}$$

We use outlier detection based on Median Absolute Deviation (MAD) [45, 46] to detect outliers in the packet transmission time. It is a robust method to detect outliers, which amplifies the effect of outliers and allows for more accurate detection of outliers from normal data.

Input: X(Time difference series)
Output: $Rate^{j}_{N_i}$(The frequency of outlier)
1: Calculate the median of X and assign it to $median$
2: Calculate the absolute value of the difference between each element in X and $median$, and assign it to $deviations$
3: Calculate the median of $deviations$ and assign it to mad
4: $outlier\_count = 0$
5: **for** i=1 to the count of $deviations$ **do**
6: **if** $|deviations_i - median| > mad \times n$ **then**
7: $outlier\_count = outlier\_count + 1$
8: **end if**
9: **end for**
10: $Rate^{j}_{N_i} = outlier\_count/count(X)$
11: **return** $Rate^{j}_{N_i}$

Algorithm 1 Outlier frequency calculation

Suppose N_i sends m packets to the Sink through path $Path^{j}_{N_i}$. The packet transmission time of these packets is $X = \{T^{N_i}_{Sink1}, T^{N_i}_{Sink2}, \ldots, T^{N_i}_{Sinkm}\}$. The MAD is calculated as follows:

$$MAD = median(abs(X - median(X))) \tag{11}$$

where $median(X)$ is the median of X, $abs(X - median(X))$ is the absolute deviation of the data points in X from the median of X.

The distance of sequence X from the $median(X)$ based on MAD is sequence $D = \{D_1, D_2, \ldots, D_i, \ldots, D_m\}$:

$$D = abs(X - median(X))/MAD \tag{12}$$

which is the absolute deviation of the data points from the median divided by the median of the absolute deviation value sequence.

D and MAD are used to determine if the data points in X are outliers.

$$\begin{cases} The\ D_i\ is\ outlier, & if\ D_i > n \times MAD \\ The\ D_i\ is\ normal, & if\ D_i \le n \times MAD \end{cases} \tag{13}$$

If $D_i > n \times MAD$, the data point is considered as an outlier and vice versa. n is a settable parameter.

For the path $Path^{j}_{N_i}$, the frequency of outlier is denoted as:

$$Rate^{j}_{N_i} = \frac{count(D_i > n \times MAD)}{count(D)} \tag{14}$$

where $count()$ is the counting function, $count(D_i > n \times MAD)$ returns the number of outliers in D, and $count(D)$ is the total number of data points in D. The pseudo code for outlier frequency calculation is shown in Algorithm 5.2.3.

For the path $Path^{j}_{N_i}$, we determine whether it is suspected based on $Rate^{j}_{N_i}$ in the historical dataset and the latest dataset. We record $Rate^{j}_{N_i}$ calculated from the historical dataset as $Rate^{j\ his}_{N_i}$. When a new detection process is performed, we fuse the new dataset with the historical dataset, and $Rate^{j}_{N_i}$ is calculated as $Rate^{j\ new}_{N_i}$.

$$\begin{cases} Path^{j}_{N_i}\ is\ suspected, & if\ Rate^{j\ new}_{N_i} > Rate^{j\ his}_{N_i} \times \alpha \\ Path^{j}_{N_i}\ is\ benign, & if\ Rate^{j\ new}_{N_i} \le Rate^{j\ his}_{N_i} \times \alpha \end{cases} \tag{15}$$

If $Rate^{j\ new}_{N_i} > Rate^{j\ his}_{N_i} \times \alpha$, $Path^{j}_{N_i}$ is considered to be suspected; otherwise, $Path^{j}_{N_i}$ is considered to be benign. α represents the maximum acceptable proportion of outlier frequency.

Due to the diversity of IoT networks routing, there are multiple routing paths between source and destination nodes, and a node may act as a relay node in multiple routing paths. If malicious nodes exist in the network, the paths containing these nodes are likely to be attacked. This means that the malicious behavior of the malicious node (forwarding packets with delay) is reflected in multiple paths. We can determine whether a node is suspected by combining the forwarding behavior of the node in multiple paths. Moreover, we use reputation values to represent the forwarding behavior of nodes. After packets are grouped according to paths, outlier detection is performed on each path group. If a path is considered benign after outlier detection, it is assigned a positive reputation value; otherwise, it is assigned a negative reputation value. A lower reputation value helps us find suspected paths, and combining multiple suspected paths helps us identify suspected nodes.

The reputation value of routing path $Path^{j}_{N_i}$ is the number of packets passing through $Path^{j}_{N_i}$, which is:

$$\begin{cases} Repu_{Path^{j}_{N_i}} = count(Pkt_{set}), & if\ Path^{j}_{N_i}\ is\ benign \\ Repu_{Path^{j}_{N_i}} = -count(Pkt_{set}), & if\ Path^{j}_{N_i}\ is\ suspected \end{cases} \tag{16}$$

where Pkt_{set} is the set of packets sent by N_i through $Path^{j}_{N_i}$, and $count(Pkt_{set})$ is the total number of packets in Pkt_{set}.

The reputation value of relay nodes in each relay path is the same as that of the path. For example, as shown in Stage 3 of Fig. 8, $Path^{1}_{N_1}$'s reputation value is $Repu_{Path^{1}_{N_1}}$. The reputation values of the relay node N_2 ($Path^{1}_{N_1}[2]$) and N_3 ($Path^{1}_{N_1}[3]$) in $Path^{1}_{N_1}$ are equal to $Repu_{Path^{1}_{N_1}}$.

5.2.4 Aggregating the reputation values of the relay nodes

We can determine whether a relay node is suspected based on its global reputation value. The global reputation value of node N_i is denoted as $Reputation_{N_i}$, and $Reputation_{N_i}$ is the sum of

the reputation values of each routing path that contains N_i as a relay node. For example, as shown in Stage 4 of Fig. 8. Node N_2 acts as a relay node in $Path^1_{N_1}$, $Path^2_{N_1}$, $Path^3_{N_1}$, and $Path^1_{N_3}$. Then the global reputation value of N_2 ($Reputation_{N_2}$) is the sum of the four paths' reputation values: $Reputation_{N_2} = Repu_{Path^1_{N_1}} + Repu_{Path^2_{N_1}} + Repu_{Path^3_{N_1}} + Repu_{Path^1_{N_3}}$

$$\begin{cases} The\ N_i\ is\ normal, & if\ Reputation_{N_i} > 0 \\ The\ N_i\ is\ suspected, & if\ Reputation_{N_i} \leq 0 \end{cases} \tag{17}$$

If $Reputation_{N_i} > 0$, N_i is considered as a benign node.

5.3 Model fusion

After filtering out the suspected nodes, it is necessary to detect whether the suspected nodes are malicious. Therefore, their packet delivery and node context information need to be collected. The Sink processes the information into datasets and then hands them over to the machine learning models for detection. To improve the performance of the detection algorithm, DNM does not use a single machine learning model for detection as in the baseline algorithm. Instead, several machine learning models that performed well in historical datasets are fused for detection.

5.3.1 Collecting packet delivery and node context information

Unlike the baseline algorithm, the nodes do not need to actively send all the information recorded locally to the Sink, which causes significant energy consumption. In DNM, only when the Sink collects the information in the nodes' $R^{tab}_{forward}$, the nodes send a part of the information recorded locally to the Sink. For example, for the suspected node N_i, N_i sends P_{id} and $V_1, V_2, \ldots, V_n$ (node context information) to the Sink. For the auxiliary node N_{i-1} of N_i, the records with N_{next} equal to N_i are filtered. P_{id} and T_{send} of these records are sent to the Sink. For the auxiliary node N_{i+1} of N_i, the records with $N_{previous}$ equal to N_i are filtered. P_{id} and $T_{receive}$ of these records are sent to the Sink. The Sink takes $V_1, V_2, ..., V_n$ as the features of the model, and $T^{N_i-1}_{N_i+1}$ calculated by $T_{receive}$ and T_{send} as the label of the model. All the obtained records are used to determine whether N_i is malicious or not.

5.3.2 Forwarding delay fusion model training and malicious node detection

Classical machine learning models include SVR, DT, Linear, Random Forest, and others. They have the advantage of low complexity and do not require a large amount of data for training. In DNM, we use multiple simple machine learning models for training and testing, and select a few that perform well for detection. We achieve higher accuracy by fusing several simple models. The pseudo code of model fusion is shown in Algorithm 2.

As shown in Fig. 9, the model fusion is divided into four stages:

1. Training process: Similar to the baseline algorithm, the Sink collects packet delivery and node context information from some stages that do not suffer from malicious attacks. For each relay path of a relay node, the above information is extracted as a historical dataset. Then it is randomly divided into a training set and a test set in a specific ratio. Multiple machine learning models are trained using the training set, and the trained models are tested by the test set, which ultimately yields the loss ($loss_{training}$) of each model for that relay path.
2. Selecting the better performing models: The models that perform better in the test set are selected from the several machine learning models trained in Stage 1.
3. Detection process: For the suspected nodes to be detected, the Sink collects packet delivery and node context information from themselves and their neighboring nodes. For each relay path of the suspected nodes, the information is processed into detection sets and then handed over to the models (picked out in Stage 2) for prediction. The loss of each model can be obtained from the predicted and actual values, denote as $loss_{detectin}$.
4. Voting for malicious nodes: A single model decides whether to vote the relay path as a malicious relay path based on the difference between $loss_{training}$ and $loss_{detectin}$. Finally, the proportion of malicious relay paths to all relay paths in the suspected node is used to decide whether the node is malicious or not.

5.4 Complexity analysis

5.4.1 The time complexity of DNM

In this section, we analyze the time complexity of DNM. Node pruning and model fusion are discussed separately.

In the node pruning stage, it is assumed that there are N sensor nodes and one Sink in the IoT network. The Sink node receives packets from M sensor nodes ($0 < M \leq N$). Assuming that the average number of packets sent by each sensor node is k ($k > 0$), the Sink node receives a total of $k \times M$ packets. When the packets' routing path is grouped, each packet's transmission time is calculated, which requires traversing all packets with the time complexity $O(k \times M)$. There are four steps to calculate the outlier frequency of each group (see Section 5.2 for details), and each step requires traversing all packets once. In summary, the time complexity of node pruning is $O(5 \times k \times M)$.

In the model fusion stage, it is assumed that there are t sub-models. The training time complexity of each sub-model is T_i ($0 < i \leq t$), and the prediction time complexity is D_i (0

Input: $Matrix_h$ (historical data matrix), $Matrix_n$ (new collected data matrix), $Regress$ (list of the regression algorithm), N (the count of the best model), $Epoch$(the times to select the best model)
Output: *benign* or *malicious*

$best\_model\_list = \{\ \}$, $history\_model\_loss = \{\ \}$
for i = 1 to $Epoch$ **do**
  A portion of $Matrix_h$ are randomly selected as $test\_set$ and remaining as the $validation\_set$
  $model\_loss = \{\ \}$
  for j=1 to the number of $Regress$ **do**
    train $Regress_j$ by $training\_set$
    get $Loss_j$ by put $test\_set$ into $model_j$
    add $model_j$,$Loss_j$ into $model\_loss$
  end for
  add $model\_loss$ into $history\_model\_loss$
  Select the N models with the lowest loss from the $model\_loss$ and add them to $best\_model\_list$
end for
$best\_models = \{\ \}$, $vote\_malicious = 0$
Select N models that appear most frequently from the best model list as $best\_models$
for i=1 to N **do**
  select a unselected model denoted as $model_i$ from $best\_models$
  get $loss_i$ by putting $Matrix_n$ into $model_i$
  if $loss_i$ > the average value of $model_i$'s $Loss_i$ in $history\_model\_loss$ **then**
    $vote\_malicious = vote\_malicious + 1$
  end if
end for
if the length of $vote\_malicious > 1/2 * N$ **then**
  return *malicious*
else
  return *benign*
end if

Algorithm 2 Model fusion detection algorithm

$< i \leq t$). For example, the training time complexity of sub-model linear regression is $O(f^2 \times n + f^3)$, and the prediction time complexity is $O(f)$, where n is the number of training samples, $n = k \times M$, and f is the number of features. The time complexity when training the model is $O(\sum_{i=1}^{t} T_i)$. From t sub-models, r $(0 < r \leq t)$ better-performing models are selected as the fusion model. The time complexity in detecting is $O(\sum_{i=1}^{r} D_i)$. Then the model fusion time complexity is $O(\sum_{i=1}^{t} T_i + \sum_{i=1}^{r} D_i)$. In summary, the time complexity of DNM is $O(5 \times k \times M + \sum_{i=1}^{t} T_i + \sum_{i=1}^{r} D_i)$, and the worst time complexity is $O(5 \times k \times N + \sum_{i=1}^{t} T_i + \sum_{i=1}^{t} D_i)$.

5.4.2 Comparison with other algorithms

We compare the time complexity of our proposed algorithm (DNM) with two other algorithms: Moussa et al. [26] and Ganesh et al. [47]. Moussa et al. calculate the offset using the information in the Report message received by the node. The whole process requires traversing through all the received packets for calculation. Assuming that the total number of packets is $k \times M$, the time complexity can be simplified to $O(k \times M)$. Ganesh et al. modeled the collected sensor data and delay values using a two-layer LSTM model. In a standard LSTM network, the time complexity of the LSTM is $O(n_c \times n_c \times 4 + n_i \times n_c \times 4 + n_c \times n_o + n_c \times 3)$ [48], where n_c is the number of memory cells, n_i is the number of input units, and n_o is the number of output units.

The time complexity of DNM is higher than Moussa et al. Because we utilize more information to model the relationship between feature values and delay values through machine learning models. Therefore, DNM has better detection performance. The learning time of the two-layer LSTM model proposed by Ganesh et al. is determined by $n_c \times (4 \times n_c + n_o)$. For tasks that require a large number of output units and memory cells, learning LSTM models become computationally expensive [48]. In contrast, the fusion model in DNM is a combination of simpler machine learning models. Generally, it has a lower time complexity than the LSTM model with complex structures.

6 Performance evaluation

This section evaluates the baseline algorithm and DNM in terms of performance and energy consumption, respectively. (1) Performance comparison evaluation: We first show the performance comparison of a single model and the fusion model. Owing to the higher detection performance of the fusion model, we used the fusion model instead of a single model in our subsequent experiments. We then explored the effects of attack strength, attack probability, malicious node percentage, and the count of nodes on different algorithms(Moussa et al. [26], Ganesh et al. [31], baseline and DNM). (2) Energy consumption evaluation: Since packet transmission occupies the major overhead of node energy consumption [49], we use the count of packets transmitted by the detection algorithms to compare energy consumption. The energy consumption of the baseline algorithm and DNM are compared before and after the attack occurs, respectively.

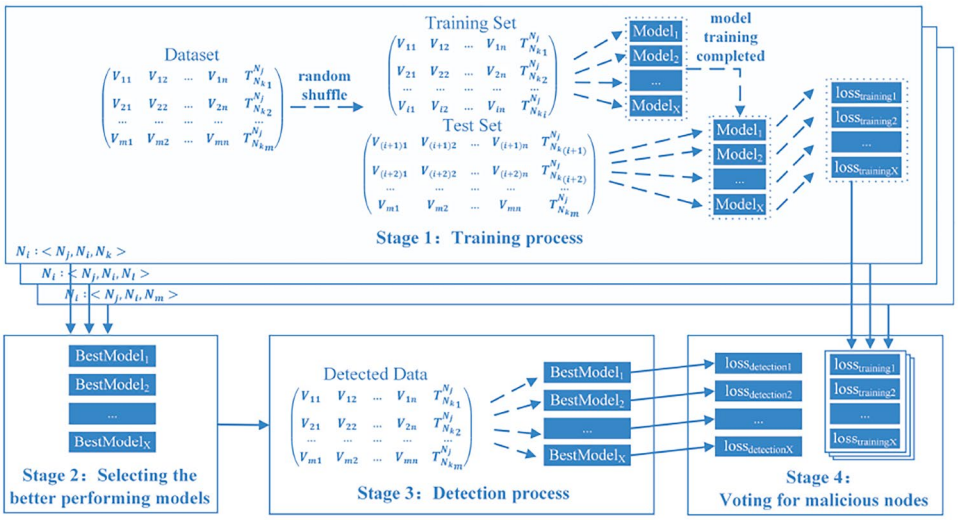

Fig. 9 Forwarding delay fusion model training and detection

6.1 Metrics

To comprehensively evaluate the performance of the algorithms, we use accuracy, false alarm rate, recall rate, and F1-score as metrics. The accuracy rate reflects the proportion of benign and malicious nodes judged by the algorithms that are actually benign and malicious nodes; the false alarm rate reflects the proportion of malicious nodes judged by the algorithms that are actually benign nodes; the recall rate reflects the proportion of benign nodes judged by the algorithms that are actually benign nodes; the F1-score reflects the overall performance of the algorithms. Based on Table 2, we define accuracy $P_a = (TP + TN)/(P + N)$, false alarm rate $F_a = FP/(TP + FP)$, recall rate $R = TP/(TP + FN)$ and F1-score $F1 = 2 \times TP/(2 \times TP + FP + FN)$.

6.2 Environment and parameter settings

As shown in Fig. 10, our experiments are simulated using the Cooja simulator [50] of the Contiki platform. The nodes are configured with the RPL protocol at the network layer for routing. The physical, link and application layers are configured using the standard protocol stack of the IoT. All IoT nodes are placed in a rectangular area of 300×300 m^2 and each node has an effective communication range of 50 m. Although the locations of the nodes in our network are randomly generated, each node is guaranteed to have at least one path to the Sink node.

To avoid bias, we simulate ten rounds for each experiment under ten network topologies. The average value is chosen as the final experimental result. We use Python and implement all algorithms using the Scikit-learn library and PyTorch. Table 3 summarizes the main variables affecting detection performance. In our experiments, we focus on their changes' effect on the algorithms' detection results.

6.3 Comparison of single model and fusion model

To show the performance difference between single model and fusion model, we conducted experiments on the baseline. As shown in Table 4, we selected 11 machine learning models for comparison. We set the upper limits of random delay time to 0.05s, 0.1s, 0.15s, 0.2s, and 0.25s, respectively, the count of nodes to 20, the percentage of malicious nodes to 0.1, and the attack probability to 0.5.

Our results are shown in Fig. 11. It can be seen that a single model does not always maintain the best performance under different attack strengths. As shown in Fig. 11, the mlp model has the highest P_a and the lowest F_a when the upper limit of the random delay time of 0.05 s. However, the lasso model has the highest P_a and the

Table 2 Experimental evaluation

	Detection result			
		Malicious	Benign	Total
Reality	Malicious	True positive (*TP*)	False negative (*FN*)	*P* (Real malicious)
	Benign	False positive (*FP*)	True negative (*TN*)	*N* (Real benign)
	Total	*P′* (Detect malicious)	*N′* (Detect begin)	*P* + *N*

Table 3 Parameter and description

Parameter	Description
Regression model	The type of machine learning model used in the baseline algorithm.
The strength of attack	The delay time of a packet when a malicious node performs the time-delay attack against a forwarded packet.
The probability of attack	The probability that a malicious node performs the time-delay attack against a forwarded packet.
The percentage of malicious nodes	The percentage of malicious nodes in the IoT network among all nodes.
The count of nodes	The count of nodes in the IoT network.

lowest F_a when the upper limit of the random delay time is increased to 0.20 s. The mlp model has the highest R when the upper limit of the random delay time is 0.05 s, while the liner and ridge models have the highest R when the delay time increases to 0.10 s. The mlp model has the highest $F1$ when the upper limit of the random delay time is 0.05s, while the lasso model has the highest $F1$ when the delay time is increased to 0.20s. Moreover, P_a, F_a, R and $F1$ of the fusion model consistently outperform most other models, especially when the attack strength is getting weaker. Because the model fusion method always selects the best-performing models in the historical data, it can well combine the excellent performance of each model on different data.

6.4 Comparison of different algorithms

Due to the lack of research on time-delay attacks in IoT, we compare baseline and DNM with the current state-of-the-art algorithms in PTP [26] and CPS [31].

For time-delay attacks in PTP, Moussa et al. [26] introduce a new PTP event message and calculate the offset using the information in the event message received by nodes. The calculated offset is expected to adhere to a pre-specified threshold, and results above the threshold indicate that nodes may suffer from time-delay attacks.

For time-delay attacks in CPS, Ganesh et al. [31] propose a deep learning-based approach to detect time-delay attacks. They design a two-layer LSTM model to process

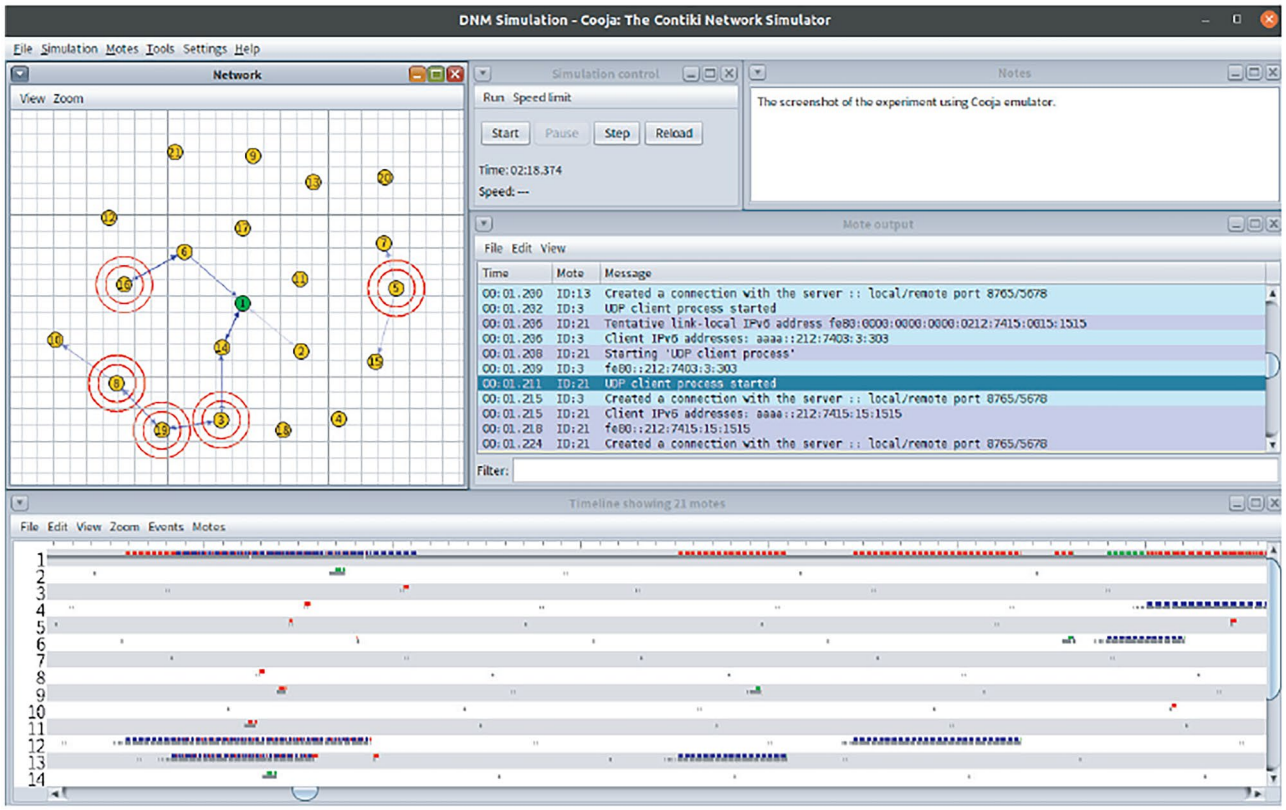

Fig. 10 The screenshot of the experiment using Cooja emulator

Table 4 Model notations

Parameter	Description
linear	Linear model
svr	Support Vector Regression model
mlp	Multilayer Perceptron model
dt	Decision Tree model
ridge	Ridge model
lasso	Lasso model
rf	Random Forest model
adaboost	Adaboost model
gbdt	Gradient Boosting Regression Tree model
bagging	Bagging model
xgb	XGBoost model
fm	Fusion model

the raw data stream from CPS sensors. The model's accuracy is measured by the difference between the predicted and actual latency, which is used to determine whether time-delay attacks exist.

6.4.1 Impact of the strength of attack

In order to evaluate the impact of attack strength on the different algorithms, we set the upper limits of random delay time to 0.05s, 0.1s, 0.15s, 0.2s, and 0.25s, respectively., the count of nodes to 20, the percentage of malicious nodes to 0.1, and the attack probability to 0.5.

The experimental results are shown in Fig. 12. As the attack strength increases, P_a, R, and $F1$ of Ganesh et al., baseline, and DNM steadily increase; F_a steadily decreases. The reason is that the malicious behavior of nodes is more obvious relative to normal nodes as the attack strength increases. Therefore these algorithms can identify malicious nodes more accurately. P_a, R, and $F1$ of Moussa et al. are low and do not vary significantly under different attack strengths, indicating that the algorithm is ineffective in detecting malicious nodes. In addition, P_a of DNM is always higher than the baseline algorithm; F_a of DNM is always lower than the baseline algorithm. The reason is that DNM only detects the suspected nodes filtered by node pruning. Some benign nodes that would cause false positives by the algorithm are not detected. $F1$ and R of DNM are always higher than the baseline, which reflects the better detection performance of DNM than the baseline.

6.4.2 Impact of the probability of attack

To evaluate the impact of attack probability, we set the attack probability as 0.1, 0.3, 0.5, 0.7, and 0.9, respectively, the count of nodes as 20, the percentage of malicious nodes as 0.1, and the upper limit of delay time as 0.15s.

As shown in Fig. 13, with the increase of attack probability, P_a, R, and $F1$ of Ganesh et al., baseline, and DNM gradually increases and then remains stable; F_a gradually decreases and then remains stable. When the attack probability is low, the node's malicious delay behavior is disguised as a normal communication delay. It can affect the arrival of packets from other benign nodes upstream of the node. The difference in behavior between malicious and benign nodes cannot be clearly distinguished, thus resulting in a higher false alarm rate. As the probability of attack increases, the malicious behavior of the nodes becomes more apparent, so these algorithms are able to identify malicious nodes more accurately. Moreover, $F1$ and P_a of DNM are always higher than the baseline algorithm; F_a of DNM is always lower than the baseline algorithm. This result reflects that EMFA's node pruning helps to identify malicious nodes more accurately.

6.4.3 Impact of the percentage of malicious nodes

In order to evaluate the impact of the percentage of malicious nodes on the different algorithms' detection performance, we set the malicious node percentage to 0.05, 0.1, 0.15, 0.2, and 0.25, respectively, the count of nodes to 20, the attack probability to 0.5, and the upper limit of delay time to 0.15s.

As shown in Fig. 14, our results demonstrate that as the percentage of malicious nodes increases, P_a, R of DNM and baseline remain basically stable, and F_a gradually decreases. We can notice that F_a of DNM is smaller than that of the baseline and other algorithms. The reason is that as the percentage of malicious nodes increases, the count of malicious nodes increases. Malicious nodes' malicious behavior affects more routing paths, which gives DNM more available path information. Combining more routing paths allows DNM to filter out suspicious nodes more accurately. Malicious node detection is performed only for these suspicious nodes, thus reducing F_a. In addition, as the percentage of malicious nodes increases, the $F1$ of DNM remains stable and higher than the baseline, reflecting that DNM can still show high robustness under different attack modes. The $F1$ of Ganesh et al. tends to increase gradually, indicating that the algorithm's overall performance increases with the number of malicious nodes.

6.4.4 Impact of the count of nodes

To evaluate the impact of nodes' count on the detection performance, we set the count of nodes to 10, 15, 20, 25, and 30, respectively. The malicious node percentage is set to 0.1, and the count of malicious nodes is set to 1, 2, 2, 3, and 3, respectively. The attack probability is set to 0.5, and the upper limit of delay time is set to 0.15s.

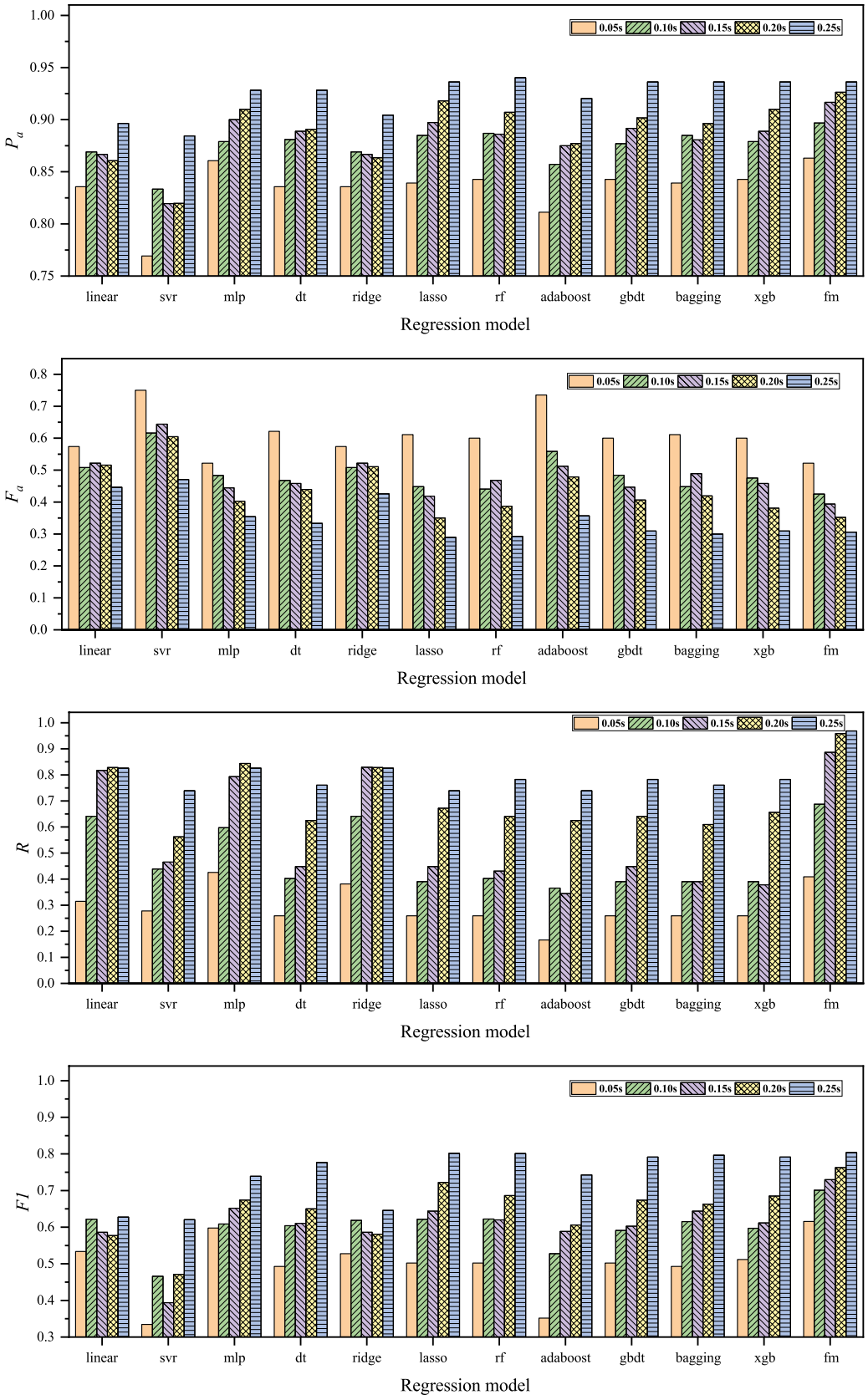

Fig. 11 The comparison of single model and model fusion in baseline algorithm

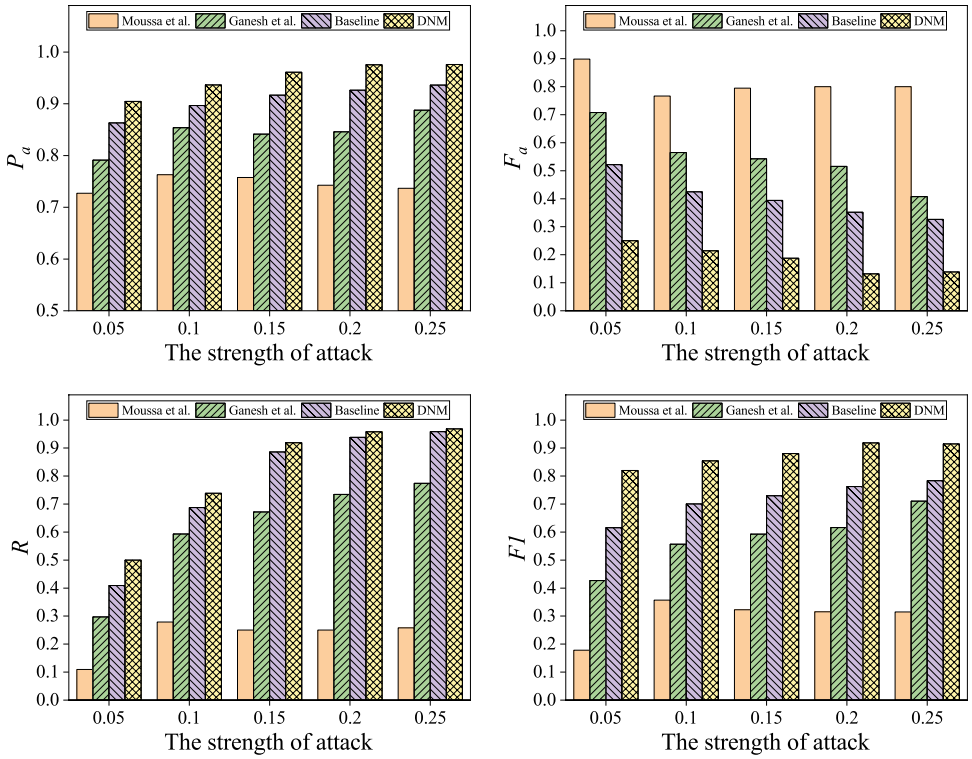

Fig. 12 The impact of the strength of attack

Our results are shown in Fig. 15: P_a, F_a, and $F1$ of the baseline algorithm and DNM remain basically stable as the count of nodes increases, indicating that both the baseline algorithm and DNM can maintain good detection performance in networks of different sizes. As the count of nodes increases, more packets are transmitted in the network. Packet collisions, retransmissions, and forwarding after waiting in the buffer queue occur more frequently during transmission. These scenarios can lead to longer packet routing times. Moreover, the larger the network size, the more

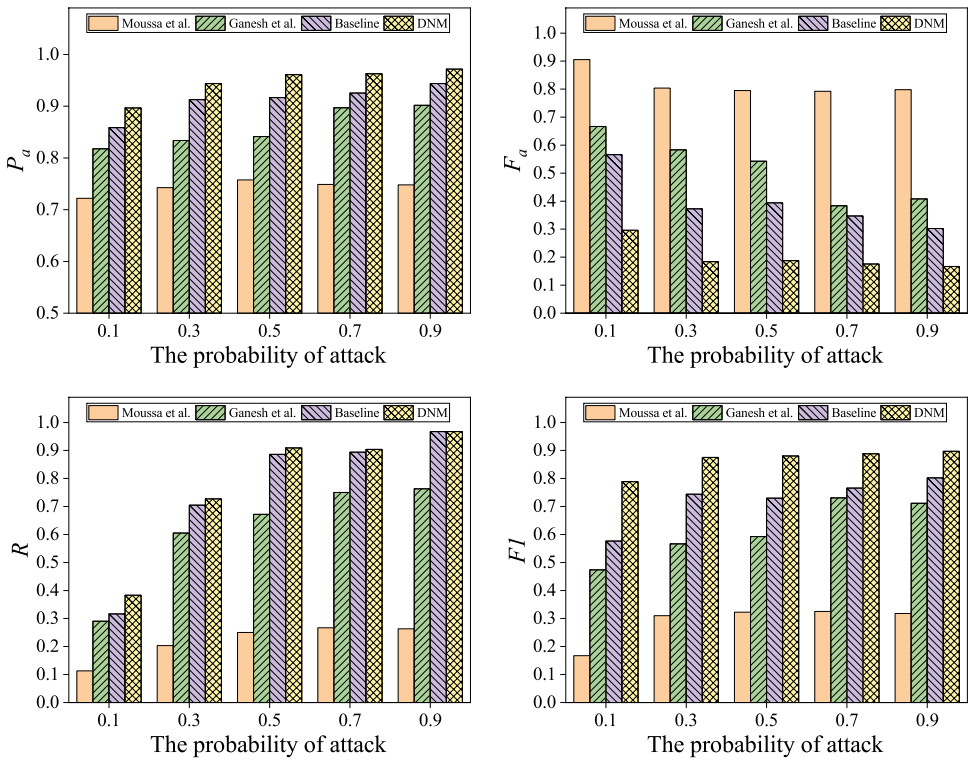

Fig. 13 The impact of the probability of attack

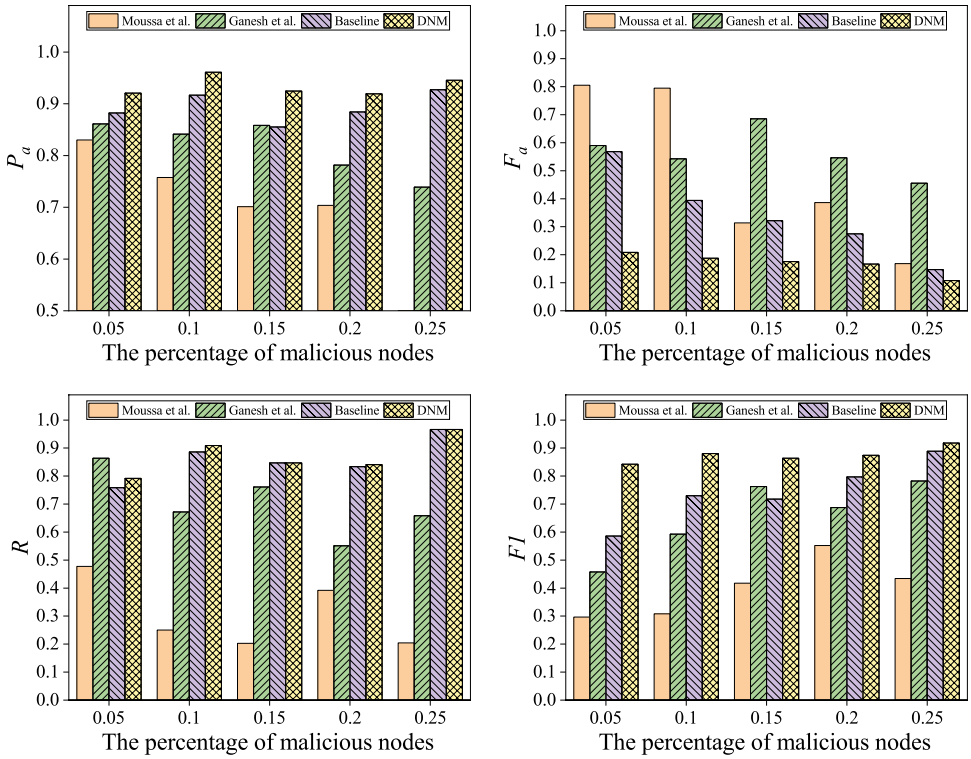

Fig. 14 The impact of the percentage of malicious nodes

complex this situation becomes, leading to a slight degradation in the algorithm's overall performance. Since Ganesh et al., baseline and DNM both take into account these factors affecting the routing time and use them as features for the training of the model. Therefore, they all show better detection performance than Moussa et al. in different node sizes.

6.4.5 The summary of performance comparison

From the above experiments in different environments, we can find that the overall detection performance of baseline and DNM is better than that of Moussa et al. and Ganesh et al. The specific reasons are as follows:

Moussa et al. [26] assume that the communication path between the master and the slave node is symmetric. In other words, the packets sent from the master node to the slave node pass through the same path as the packets from the slave node to the master node. However, this assumption does not hold true in multi-hop IoT. Because there may be multiple paths between two nodes in the multi-hop IoT, packets are selected for the best route based on the routing protocol. Therefore, the packet transmission path between two nodes may change due to network topology and environment changes. Moreover, since the network structure of PTP is relatively simple, they only consider the time characteristics. In contrast, the network structure of multi-hop IoT is complex, and the forwarding delay of packets is affected by many factors (e.g., the times of packet collisions, the time that packets wait in the buffer, etc.). It is difficult to determine whether a node is under attack by considering only the time characteristic.

Ganesh et al. [31] utilize a two-layer LSTM model with a complex network structure to model the relationship between delay values and various features. Thus, their detection method performs better than Moussa et al. and is close to baseline (modeling using machine learning). In CPS, most packets are transmitted between the fixed controller and the actuators (sensors). The packet transmission path between two nodes is fixed, and the packet transmission is continuous. Therefore there may be an implicit delay pattern, which can be learned using the LSTM model with forgetting, input, and output gates. While in multi-hop IoT, the path of packet transmission is variable, and the nodes receive discontinuous packets from different nodes due to the changing network environment. This situation is common in multi-hop IoT. Therefore it is difficult for the LSTM model to learn the implicit delay patterns in multi-hop IoT.

The detection performance of baseline and DNM is generally similar. However, DNM filters out suspected nodes and performs detection only for suspected nodes. Since DNM uses node pruning to remove nodes that may cause false positives in advance, DNM has a lower F_a and higher R than the baseline in different environments. Therefore DNM has a higher $F1$ score. In addition, node pruning significantly reduces the algorithm's detection overhead, as demonstrated in the subsequent experiments.

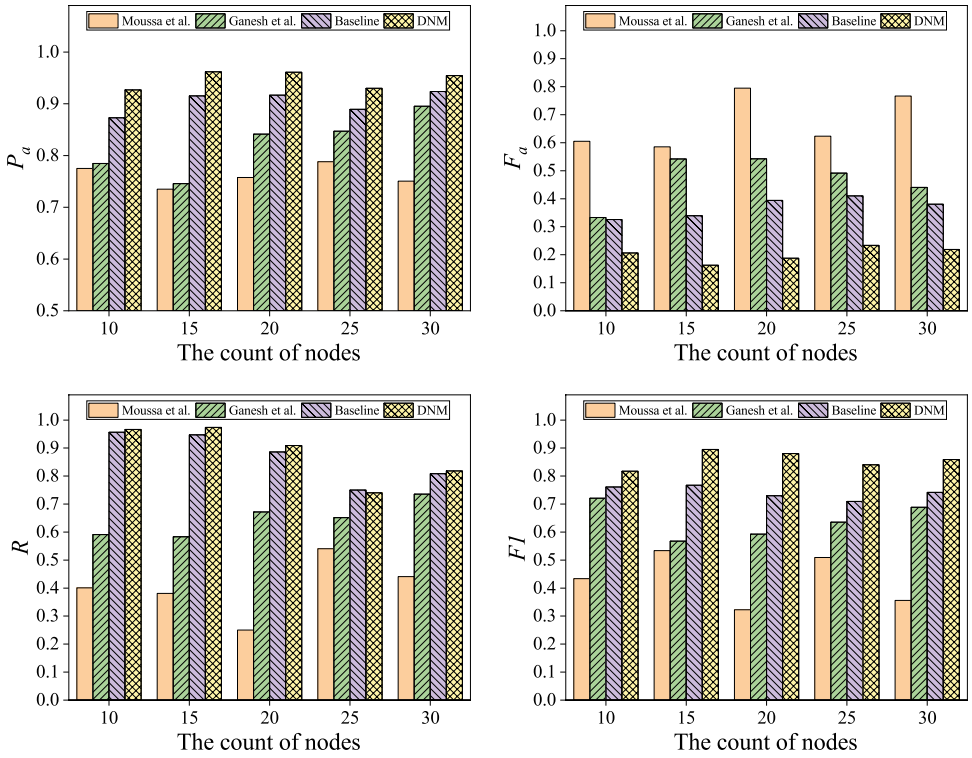

Fig. 15 The impact of the count of nodes

6.5 Comparison of energy consumption for the baseline algorithm and DNM

We compare the extra energy consumption of the baseline algorithm and DNM from two perspectives, before and after the attack occurs.

6.5.1 Comparison of detection packets collected before the attack

We compared the count of detection packets required to be transmitted by the baseline algorithm and DNM in the same time when no attack occurs, which characterizes the energy consumption of the algorithms under normal conditions. We set the upper limit of the random delay time to 0.15s, the count of nodes to 20, the percentage of malicious nodes to 0.1, and the attack probability to 0.5.

Our results are shown in Fig. 16. According to obtained results, we can find that as the count of nodes in the network increases, the count of detection packets transmitted by the baseline algorithm and DNM shows an incremental trend. However, the count of detection packets transmitted by DNM is much smaller than that of the baseline algorithm. In the case of 30 nodes, the count of detection packets DNM needs to transmit is only about 7% of the baseline algorithm. This reason is that the baseline algorithm detects all non-leaf nodes, and each node sends all the data recorded by it. In contrast, DNM detects only the suspected nodes through node pruning, which significantly reduces the transmission of packets required for algorithm detection. Moreover, node pruning is based on PPPT, and the extra overhead brought by PPPT is negligible compared to the energy consumed by normal packet transmission [44]. Therefore, DNM achieves low energy consumption overhead compared to the baseline algorithm by node pruning.

6.5.2 Comparison of detection packets collected after the attack

We compared the amount of detection packets required to be transmitted by the baseline algorithm and DNM after the attack occurred. We set the upper limit of the random delay

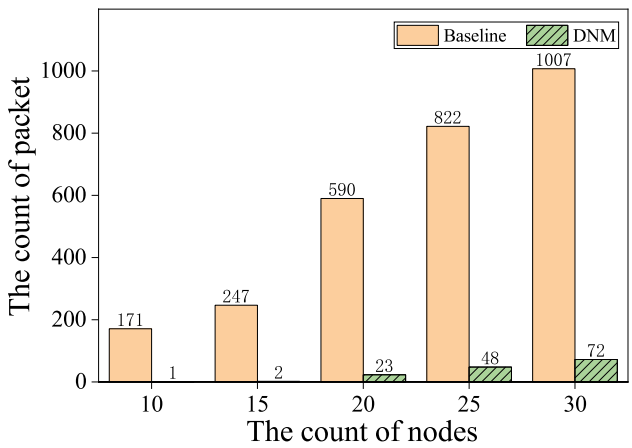

Fig. 16 The count of nodes detected before the attack

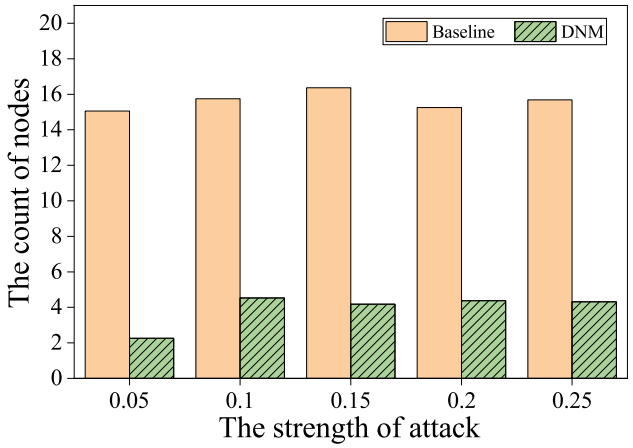

Fig. 17 The count of nodes detected after the attack

time to 0.15s, the count of nodes to 20, the percentage of malicious nodes to 0.1, and the attack probability to 0.5.

The count of nodes detected by the baseline algorithm and DNM for different attack strengths is illustrated in Fig. 17. It can be seen that the count of nodes to be detected in the baseline algorithm is stable at around 16, which is the count of non-leaf nodes in the network involved in the packet forwarding process. The count of nodes to be detected in DNM is stable at around 4 due to the node pruning mechanism, which dramatically reduces the count of nodes to be detected compared to the baseline algorithm.

As shown in Fig. 18, it shows that the baseline algorithm and DNM at different attack strengths: (1) The ratio of the count of nodes detected; (2) The ratio of the count of additional packets to be transmitted for detection. As the attack strength increases, the detection performance of both algorithms tends to stabilize, and the ratios also tend to stabilize. The count of detection packets required by DNM is only about 15% of the baseline algorithm's requirements. The ratio of packets required by DNM is significantly smaller than that of detected nodes because the baseline algorithm collects all the data stored in the nodes. In contrast, DNM only collects some of the data stored in the nodes. These data are necessary to be used directly to detect malicious nodes.

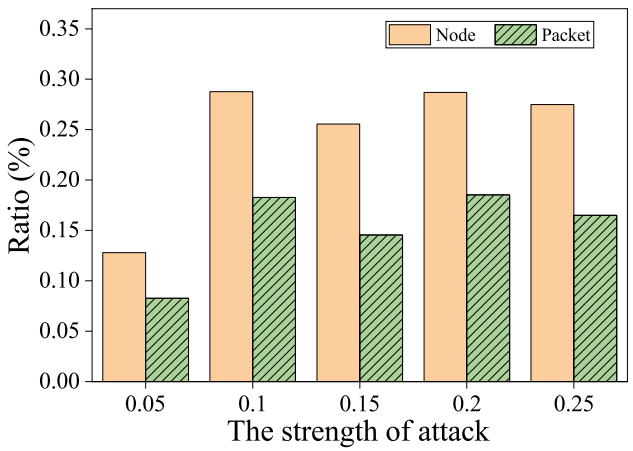

Fig. 18 The ratio of DNM to baseline algorithm after the attack: the count of detected nodes and the count of additional detected packets transmitted

7 Conclusion and future work

The detection of time-delay attacks in IoT networks is an open problem. In this paper, we propose the baseline algorithm and DNM, respectively. The baseline algorithm builds machine learning models for all nodes without selectivity, which leads to significant energy consumption. In contrast, DNM relies on node pruning to detect only suspected nodes, thus reducing energy consumption. Moreover, DNM also achieves higher detection performance than the baseline algorithm by model fusion. The experimental results show that both the baseline algorithm and DNM have good detection performance. DNM has a higher detection performance and smaller overhead than the baseline algorithm.

The overhead of our detection algorithms relies on the forwarding features recorded by the nodes, and sending these features to the Sink for centralized analysis is a non-negligible overhead. Therefore, how to use as few features as possible or fuse features would be an optimization direction. Furthermore, various attack variants of the time-delay attack would be further investigated, such as selective delayed packets and delayed ack message packet delivery. These attack variants are more difficult to detect. As well as the hybrid attacks combining time-delay attacks with other attacks would also be a future research direction.

Author contributions Wenjie Zhao: Conceptualization, Data curation, Software, Formal analysis, Methodology, Writing - original draft, Writing - review & editing. Yu Wang: Investigation, Methodology, Software, Writing - original draft. Wenbin Zhai: Conceptualization, Resources, Funding acquisition, Project administration, Supervision, Writing - review & editing. Liang Liu: Methodology, Formal analysis, Supervision, Writing - review & editing. Yulei Liu: Writing - review & editing.

Funding This work is supported by the National Key R &D Program of China under No. 2021YFB2700500 and 2021YFB2700502, the Open Fund of Key Laboratory of Civil Aviation Smart Airport Theory and System, Civil Aviation University of China under No. SATS202206, the National Natural Science Foundation of China under No. U20B2050, Public Service Platform for Basic Software and Hardware Supply Chain Guarantee under No. TC210804A.

Data availability The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Ethics approval Not applicable.

Consent to publish All of the authors have approved the contents of this paper and have agreed to the submission policies of Peer-to-Peer Networking and Applications.

Conflict of interest We declare that we have no competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Aheleroff S, Xu X, Lu Y, Aristizabal M, Velásquez JP, Joa B, Valencia Y (2020) Iot-enabled smart appliances under industry 4.0: A case study. Adv Eng Inform 43. https://doi.org/10.1016/j.aei.2020.101043
2. Viswanath SK, Yuen C, Tushar W, Li W-T, Wen C-K, Hu K, Chen C, Liu X (2016) System design of the internet of things for residential smart grid. IEEE Wirel Commun 23(5):90–98. https://doi.org/10.1109/MWC.2016.7721747
3. Fang S, Da Xu L, Zhu Y, Ahati J, Pei H, Yan J, Liu Z (2014) An integrated system for regional environmental monitoring and management based on internet of things. IEEE Trans Industr Inf 10(2):1596–1605. https://doi.org/10.1109/TII.2014.2302638
4. Wang D, Chen D, Song B, Guizani N, Yu X, Du X (2018) From iot to 5g i-iot: The next generation iot-based intelligent algorithms and 5g technologies. IEEE Commun Mag 56(10):114–120. https://doi.org/10.1109/MCOM.2018.1701310
5. Pokhrel SR, Vu HL, Cricenti AL (2019) Adaptive admission control for iot applications in home wifi networks. IEEE Trans Mob Comput 19(12):2731–2742. https://doi.org/10.1109/TMC.2019.2935719
6. Li Y, Chi Z, Liu X, Zhu T (2018). Passive-zigbee: Enabling zigbee communication in iot networks with 1000x+ less power consumption. In: Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems, pp. 159–171. https://doi.org/10.1145/3274783.3274846
7. Kim H-S, Ko J, Culler DE, Paek J (2017) Challenging the ipv6 routing protocol for low-power and lossy networks (rpl): A survey. IEEE Commun Surv Tutorials 19(4):2502–2525. https://doi.org/10.1109/COMST.2017.2751617
8. Deogirikar J, Vidhate A (2017) Security attacks in iot: A survey. In: 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), pp. 32–37. https://doi.org/10.1109/I-SMAC.2017.8058363
9. Stellios I, Kotzanikolaou P, Psarakis M, Alcaraz C, Lopez J (2018) A survey of iot-enabled cyberattacks: Assessing attack paths to critical infrastructures and services. IEEE Commun Surv Tutorials 20(4):3453–3495. https://doi.org/10.1109/COMST.2018.2855563
10. Prathapchandran K, Janani T (2021) A trust aware security mechanism to detect sinkhole attack in rpl-based iot environment using random forest-rftrust. Comput Netw 198:108413. https://doi.org/10.1016/j.comnet.2021.108413
11. Divya K, Jaipriya S, Anitha G, Malathy S, Maheswar R (2018) An energy efficient technique for time sensitive application using mc-wsn. In: 2018 2nd International Conference on Inventive Systems and Control (ICISC), pp. 1451–1455. https://doi.org/10.1109/ICISC.2018.8399048
12. Poe WY, Schmitt JB (2008) Placing multiple sinks in time-sensitive wireless sensor networks using a genetic algorithm. In: 14th GI/ITG Conference-Measurement, Modelling and Evalutation of Computer and Communication Systems, pp. 1–15
13. Korala H, Georgakopoulos D, Jayaraman PP, Yavari A (2022) A survey of techniques for fulfilling the time-bound requirements of time-sensitive iot applications. ACM Comput Surv. https://doi.org/10.1145/3510411
14. Song H, Zhu S, Cao G (2007) Attack-resilient time synchronization for wireless sensor networks. Ad Hoc Netw 5(1):112–125. https://doi.org/10.1016/j.adhoc.2006.05.016
15. Lee JH, Shin J, Realff MJ (2018) Machine learning: Overview of the recent progresses and implications for the process systems engineering field. Comput Chem Eng 114:111–121. https://doi.org/10.1016/j.compchemeng.2017.10.008
16. Chen Z, Liu J, Shen Y, Simsek M, Kantarci B, Mouftah HT, Djukic P (2022) Machine learning-enabled iot security: Open issues and challenges under advanced persistent threats. ACM Comput Surv 55(5):1–37. https://doi.org/10.1145/3530812
17. Huang X, Wu Y (2022) Identify selective forwarding attacks using danger model: Promote the detection accuracy in wireless sensor networks. IEEE Sens J 22(10):9997–10008. https://doi.org/10.1109/JSEN.2022.3166601
18. Ding J, Wang H, Wu Y (2022) The detection scheme against selective forwarding of smart malicious nodes with reinforcement learning in wireless sensor networks. IEEE Sens J 22(13):13696–13706. https://doi.org/10.1109/JSEN.2022.3176462
19. Chen X, Feng W, Luo Y, Shen M, Ge N, Wang X (2022) Defending against link flooding attacks in internet of things: A bayesian game approach. IEEE Internet Things J 9(1):117–128. https://doi.org/10.1109/JIOT.2021.3093538
20. Srinivas TAS, Manivannan S (2020) Prevention of hello flood attack in iot using combination of deep learning with improved rider optimization algorithm. Comput Commun 163:162–175. https://doi.org/10.1016/j.comcom.2020.03.031
21. Teng Z, Du C, Li M, Zhang H, Zhu W (2022) A wormhole attack detection algorithm integrated with the node trust optimization model in wsns. IEEE Sens J 22(7):7361–7370. https://doi.org/10.1109/JSEN.2022.3152841
22. Pu C, Choo K-KR (2022) Lightweight sybil attack detection in iot based on bloom filter and physical unclonable function. Comput Secur 113:102541. https://doi.org/10.1016/j.cose.2021.102541
23. Alghamdi R, Bellaiche M (2023) A cascaded federated deep learning based framework for detecting wormhole attacks in iot networks. Comput Secur 125:103014. https://doi.org/10.1016/j.cose.2022.103014
24. Kim J-D, Ko M, Chung J-M (2022) Physical identification based trust path routing against sybil attacks on rpl in iot networks. IEEE Wireless Commun Lett 11(5):1102–1106. https://doi.org/10.1109/LWC.2022.3157831
25. Moradi M, Jahangir AH (2021) A new delay attack detection algorithm for ptp network in power substation. Int J Electr Power Energy Syst 133:107226. https://doi.org/10.1016/j.ijepes.2021.107226
26. Moussa B, Kassouf M, Hadjidj R, Debbabi M, Assi C (2020) An extension to the precision time protocol (ptp) to enable the detection of cyber attacks. IEEE Trans Industr Inf 16(1):18–27. https://doi.org/10.1109/TII.2019.2943913
27. Wang J, Peng C (2017) Analysis of time delay attacks against power grid stability. In: Proceedings of the 2nd Workshop on Cyber-Physical Security and Resilience in Smart Grids, pp. 67–72. https://doi.org/10.1145/3055386.3055392
28. De Pace G, Wang Z, Benin J, He H, Sun Y (2020) Evaluation of communication delay based attack against the smart grid. In: 2020 IEEE Kansas Power and Energy Conference (KPEC), pp. 1–6. https://doi.org/10.1109/KPEC47870.2020.9167543
29. Lou X, Tran, C, Yau DK, Tan R, Ng H, Fu, TZ, Winslett M (2019) Learning-based time delay attack characterization for cyber-physical systems. In: 2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), pp. 1–6 . https://doi.org/10.1109/SmartGridComm.2019.8909732

30. Abbasspour A, Sargolzaei A, Victorio M, Khoshavi N (2020) A neural network-based approach for detection of time delay switch attack on networked control systems. Procedia Computer Science 168:279–288. https://doi.org/10.1016/j.procs.2020.02.250
31. Ganesh P, Lou X, Chen Y, Tan R, Yau DKY, Chen D, Winslett M (2021) Learning-based simultaneous detection and characterization of time delay attack in cyber-physical systems. IEEE Trans Smart Grid 12(4):3581–3593. https://doi.org/10.1109/TSG.2021.3058682
32. Sargolzaei A, Yen KK, Abdelghani MN (2015) Preventing time-delay switch attack on load frequency control in distributed power systems. IEEE Trans Smart Grid 7(2):1176–1185. https://doi.org/10.1109/TSG.2015.2503429
33. Victorio M, Sargolzaei A, Khalghani MR (2021) A secure control design for networked control systems with linear dynamics under a time-delay switch attack. Electronics 10(3):322. https://doi.org/10.3390/electronics10030322
34. Altaf A, Abbas H, Iqbal F, Khan MMZM, Rauf A, Kanwal T (2021) Mitigating service-oriented attacks using context-based trust for smart cities in iot networks. J Syst Archit 115:102028. https://doi.org/10.1016/j.sysarc.2021.102028
35. Mabodi K, Yusefi M, Zandiyan S, Irankhah L, Fotohi R (2020) Multi-level trust-based intelligence schema for securing of internet of things (iot) against security threats using cryptographic authentication. J Supercomput 76(9):7081–7106. https://doi.org/10.1007/s11227-019-03137-5
36. Liu L, Ma Z, Meng W (2019) Detection of multiple-mix-attack malicious nodes using perceptron-based trust in iot networks. Futur Gener Comput Syst 101:865–879. https://doi.org/10.1016/j.future.2019.07.021
37. Liu L, Xu X, Liu Y, Ma Z, Peng J (2021) A detection framework against cpma attack based on trust evaluation and machine learning in iot network. IEEE Internet Things J 8(20):15249–15258. https://doi.org/10.1109/JIOT.2020.3047642
38. Ma Z, Liu L, Meng W (2020) Towards multiple-mix-attack detection via consensus-based trust management in iot networks. Comput Secur 96:101898. https://doi.org/10.1016/j.cose.2020.101898
39. Singh M, Sardar AR, Majumder K, Sarkar SK (2017) A lightweight trust mechanism and overhead analysis for clustered wsn. IETE J Res 63(3):297–308. https://doi.org/10.1080/03772063.2017.1284613
40. Poongodi T, Khan MS, Patan R, Gandomi AH, Balusamy B (2019) Robust defense scheme against selective drop attack in wireless ad hoc networks. IEEE Access 7:18409–18419. https://doi.org/10.1109/ACCESS.2019.2896001
41. Eskandari M, Janjua ZH, Vecchio M, Antonelli F (2020) Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. IEEE Internet Things J 7(8):6882–6897. https://doi.org/10.1109/JIOT.2020.2970501
42. Nguyen TD, Marchal, S, Miettinen M, Fereidooni H, Asokan N, Sadeghi AR (2019) Dïot: A federated self-learning anomaly detection system for iot. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 756–767. https://doi.org/10.1109/ICDCS.2019.00080
43. Moussa B, Debbabi M, Assi C (2016) A detection and mitigation model for ptp delay attack in an iec 61850 substation. IEEE Trans Smart Grid 9(5):3954–3965. https://doi.org/10.1109/TSG.2016.2644618
44. Suhail S, Hussain R, Abdellatif M, Pandey SR, Khan A, Hong CS (2020) Provenance-enabled packet path tracing in the rpl-based internet of things. Comput Netw 173:107189. https://doi.org/10.1016/j.comnet.2020.107189
45. Rousseeuw PJ, Croux C (1993) Alternatives to the median absolute deviation. J Am Stat Assoc 88(424):1273–1283. https://doi.org/10.1080/01621459.1993.10476408
46. Chen Z, Song S, Wei Z, Fang J, Long J (2021) Approximating median absolute deviation with bounded error. Proceedings of the VLDB Endowment 14(11):2114–2126. https://doi.org/10.14778/3476249.3476266
47. Ganesh P, Lou X, Chen Y, Tan R, Yau DK, Chen D, Winslett M (2021) Learning-based simultaneous detection and characterization of time delay attack in cyber-physical systems. IEEE Trans Smart Grid 12(4):3581–3593. https://doi.org/10.1109/TSG.2021.3058682
48. Sak H, Senior AW, Beaufays F (2014) Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In: INTERSPEECH, pp. 338–342
49. Ganti RK, Jayachandran P, Luo H, Abdelzaher TF (2006) Datalink streaming in wireless sensor networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, pp. 209–222. http://doi.org/10.1145/1182807.1182829
50. Osterlind F, Dunkels A, Eriksson, J, Finne N, Voigt T (2006) Cross-level sensor network simulation with cooja. In: Proceedings. 2006 31st IEEE Conference on Local Computer Networks, pp. 641–648. https://doi.org/10.1109/LCN.2006.322172