Deep Learning-Guided Jamming for Cross-Technology Wireless Networks: Attack and Defense

Dianqi Han, *Graduate Student Member, IEEE*, Ang Li, Lili Zhang, Yan Zhang, *Graduate Student Member, IEEE*, Jiawei Li, *Graduate Student Member, IEEE*, Tao Li, *Member, IEEE*, Ting Zhu, *Senior Member, IEEE*, and Yanchao Zhang, *Fellow, IEEE*

***Abstract*— Wireless networks of different technologies may interfere with each other when they are deployed at proximity. Such cross-technology interference (CTI) has become prevalent with the surge of IoT devices. In this paper, we exploit CTI in coexisting WiFi-Zigbee networks and propose DeepJam, a new stealthy jamming strategy, to jam Zigbee traffic. DeepJam relies on deep learning techniques to capture the temporal pattern of the past wireless traffic and predict the future wireless traffic. By only jamming the victim's transmissions that are not disrupted by CTI, DeepJam can significantly reduce the victim's throughput with far fewer jamming signals and is thus much more stealthy than conventional jamming strategies. Detailed evaluations show that DeepJam can converge within 10 sec and achieve the jamming-efficiency gains of up to 742% and 285% over conventional random and reactive jamming strategies, respectively, in practical scenarios. We also propose a simple yet effective countermeasure against DeepJam.**

***Index Terms*— Jamming, cross-technology interference, WiFi and Zigbee, reinforcement learning.**

I. Introduction

JAMMING is a critical threat against wireless communications. Different jamming attacks have been proposed for attackers with various capabilities [1]–[3], and there have also been many studies on defeating or detecting jamming attacks [4], [5]. The current wireless environment has become much more complex than before due to the surge of IoT devices all over the world, which results in new threats to wireless security as shown in previous studies [6]. The emerging deep learning technique has demonstrated its efficacy in compromising wireless security [7], [8]. It is thus meaningful to investigate whether the attacker can exploit the complex wireless environment and deep learning technique to launch more effective jamming attacks and also devise corresponding countermeasures.

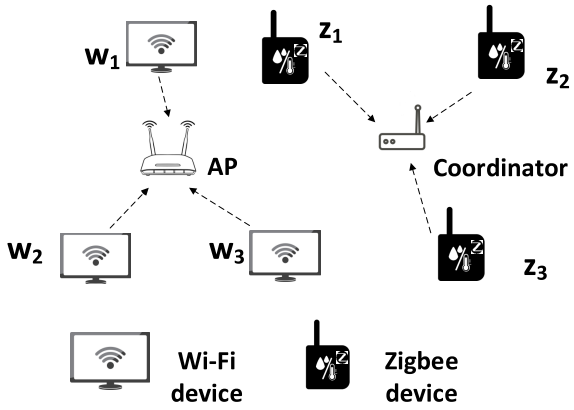


Fig. 1. Targeted CTI context of DeepJam.

The unlicensed frequency bands are now crowded with devices of different wireless technologies. For example, WiFi, Zigbee, and Bluetooth all use the 2.4 GHz ISM band. The Cross-Technology Interference (CTI) happens when different kinds of wireless networks on the same frequency band are deployed at proximity. For instance, previous work shows that the throughput of a Zigbee network may drop by more than 60% if a WiFi network coexists [9], [10].

This paper proposes DeepJam, a new deep learning-guided jamming attack that exploits CTI in the complex wireless environment. We illustrate the basic idea of DeepJam with an example CTI context shown in Fig. 1, where a WiFi network and a Zigbee network coexist. We consider this context because of the prevalence of WiFi and Zigbee networks, and our work can be extended to other scenarios in which CTI exists. The WiFi network contains one Access Point (AP) and multiple WiFi devices, which can be mobile devices,

Manuscript received January 4, 2021; revised February 17, 2021; accepted March 8, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Chen. Date of publication May 28, 2021; date of current version October 15, 2021. This work was supported in part by the U.S. National Science Foundation under Grant CNS 1619251, Grant 1652669, Grant 1824355, Grant 1824491, and Grant 1933069. *(Corresponding author: Yanchao Zhang.)*

Dianqi Han, Ang Li, Lili Zhang, Yan Zhang, Jiawei Li, and Yanchao Zhang are with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: dqhan@asu.edu; anglee@asu.edu; lilizhang@asu.edu; yanzhangyz@asu.edu; jwli@asu.edu; yczhang@asu.edu).

Tao Li is with the Computer and Information Technology Department, Indiana University-Purdue University Indianapolis, Indianapolis, IN 46202 USA (e-mail: tli6@iupui.edu).

Ting Zhu is with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250 USA (e-mail: zt@umbc.edu).

Digital Object Identifier 10.1109/TNET.2021.3082839

computers, or smart home devices. The Zigbee network contains one coordinator and multiple Zigbee devices, and the network can be the alarm system in a house or the temperature monitoring system in a factory. The adversary aims to disrupt the Zigbee traffic from a specific device in an efficient and stealthy manner. In this context, the victim's transmission may fail with a high probability when CTI happens. Therefore, reactive jamming, which generates a jamming signal upon detection of any Zigbee preamble, is inefficient because it wastes significant energy jamming the victim's transmissions already disrupted by CTI and also the transmissions of other Zigbee devices. Random jamming that transmits a jamming signal regardless of the presence or absence of the victim's transmissions is not only more energy-inefficient but also easier to detect. By comparison, DeepJam can significantly reduce the victim's throughput with far fewer jamming signals by only jamming the victim's transmissions which are not subject to CTI and thus achieve much more stealthy jamming than conventional random and reactive jamming strategies.

It is challenging to accurately predict when jamming is necessary. The random backoff periods and the asynchronous clocks of the WiFi and Zigbee networks result in chaotic wireless traffic, making it difficult to capture the temporal traffic patterns. We first investigate the Zigbee MAC protocol and propose a slotted formulation of the jamming attack. By properly choosing the slot duration and carefully defining the status and actions, we convert the jamming attack to a time series process. Then we propose a deep learning-guided strategy to predict the attacker's optimal action in the coming slot according to sniffed traffic in the past. Deep learning has been proven more efficient in solving time series problems than traditional machine learning methods and thus is more suitable for DeepJam.

We evaluate the performance of DeepJam with comprehensive comparison with random jamming and reactive jamming in different scenarios. The results show that DeepJam significantly outperforms random jamming in all scenarios. Although reactive jamming performs better than DeepJam in a simple system with one Zigbee device, DeepJam outperforms reactive jamming in terms of efficiency and stealth when there are multiple Zigbee devices, and its advantage becomes more significant with the increase of the network complexity. For example, in a system that contains five Zigbee devices, 42% of DeepJam's jamming actions are necessary, while only 13% of reactive jamming actions are necessary. Due to the random backoff mechanism of the Zigbee network, DeepJam cannot fully jam the traffic of the victim device. But we show that DeepJam can decrease the throughput of the victim by 60% in the worst case and by 78% in the best case. We also propose a simple countermeasure against DeepJam and evaluated its efficacy.

The rest of the paper is organized as follows. Section II reviews the background of Zigbee and CTI. Section III presents the system and adversary models. Section IV gives the problem formulation. Sections V illustrates the technical details of DeepJam. Section VI discusses two countermeasures. Section VII outlines the simulation process for Zigbee and WiFi networks. Section VIII evaluates DeepJam with comparison to random jamming and reactive jamming. Section IX briefs the related work. Section X concludes our work.

II. Background

A. Zigbee MAC Layer

The Zigbee MAC layer is defined in IEEE 802.15.4. The network can be beacon-disabled or beacon-enabled. The former adopts unslotted Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) for media access control, and the latter adopts slotted CSMA/CA. The acknowledgment (ACK) is optional in Zigbee MAC layer. In a beacon-disabled Zigbee network, a device first conducts the Clear Channel Assessment (CCA) to determine the channel status when it attempts to transmit a MAC frame. The device immediately transmits the MAC frame when the channel is idle. Otherwise, the device waits for a period and tries again. The backoff time B_z is defined by an exponential backoff algorithm. In a beacon-enabled network, the coordinator periodically generates beacons which divide the channel to superframes of the same duration. A superframe contains a mandatory active period and an optional inactive period, and the active period is equally divided into 16 slots. During the active period, a device accesses the channel in a similar manner as in the beacon-disabled Zigbee network, while the frame transmission must start at the beginning of one slot.

A Zigbee network can adopt Energy Detection (ED) or Carrier Sensing (CS) for CCA. ED considers the channel busy if the energy level is above a predefined threshold, while CS considers the channel busy only if Zigbee signals are detected. Payload encryption is optional in Zigbee MAC layer, and the Frame Check Sum (FCS) occupying the last 2 bytes of a MAC frame is used to detect bit errors. The MAC frame header, which contains the source and destination MAC addresses, and FCS are transmitted in plaintext.

B. Cross-Technology Interference (CTI) Illustration

As shown in Fig. 2, the Zigbee and WiFi channels overlap in frequency. For example, the WiFi channel 6 uses the same frequency band as Zigbee channels 16-19. Consequently, a WiFi network working on channel 6 may interfere with a Zigbee network working on channel 18 if those two networks are deployed at proximity.

Previous studies have experimentally evaluated the interference between WiFi and Zigbee networks [9], [10]. Since a WiFi transmitter's power is more than 30 times larger than a Zigbee transmitter's, Zigbee signals' impact on WiFi transmissions is insignificant. In contrast, WiFi signals have significant impact on Zigbee transmissions. Although WiFi signals do not fully jam Zigbee signals, the throughput of the Zigbee network may drop by 60% due to CTI.

III. System and Adversary Models

A. System Model

We consider a system in which a Zigbee network and a WiFi network coexist, and the system model is shown in Fig. 1. The experimental result in [9] shows that the

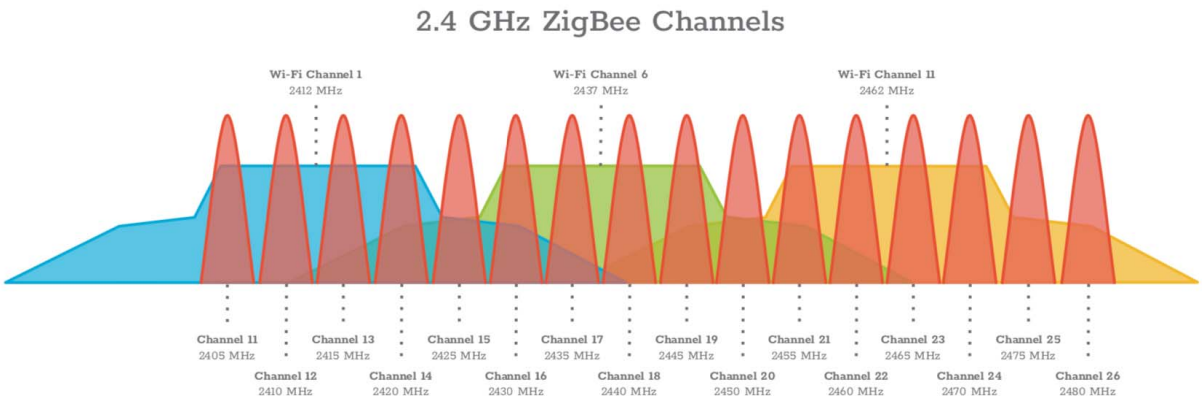


Fig. 2. Zigbee and WiFi channels.

cross-technology interference between Zigbee and Bluetooth networks is insignificant, so we do not consider Bluetooth networks in our system. The WiFi network follows IEEE 802.11 and contains one AP and multiple WiFi devices. The Zigbee network follows IEEE 802.15.4 and contains one coordinator and multiple Zigbee devices. Without loss of generality, we assume that the WiFi network keeps working on WiFi channel 6 (with a center frequency of 2,437 MHz and bandwidth of 20 MHz), and the Zigbee network keeps working on channel 18 (with a center frequency of 2,440 MHz and a bandwidth of 2 MHz). In this case, the WiFi network interferes with the Zigbee network. IEEE 802.15.4 does not adopt frequency hopping, so the Zigbee network does not hop to another channel due to the interference. Although the WiFi network adopts channel hopping, the Zigbee signal's power is too low to cause significant interference to the WiFi network which thus does not change the channel either.

We make the following assumptions about the Zigbee network. First, the Zigbee network is busy and all the MAC frames are of the maximum length $\mathcal{L}_z$ to maximize the data transmission rate. $\mathcal{L}_z$ equals 127 bytes as defined in 802.15.4. Second, we assume that the network's traffic load and pattern are both stable within a short period (less than 1 min). For simplicity, we assume that the number of Zigbee MAC frames generated (not transmitted) by a Zigbee device within a short period follows the Poisson distribution, and the possibility that a device generates k_z new MAC frames within τ sec is formulated as

$$P(k_z) = \frac{e^{-\lambda_z \tau}(\lambda_z \tau)^{k_z}}{k_z!}, \tag{1}$$

where λ_z is the arrival rate. The arrival rate varies with the device's ongoing task, so we only assume that λ_z is stable within a short period.

B. Adversary Model

We consider an attacker who is aware of the Zigbee channel by passive eavesdropping and attempts to disrupt the traffic from a specific Zigbee device in a stealthy manner. Particularly, the objective of the attacker is to reduce the throughput of the victim device by a target percentage with as few jamming signals as possible. Without loss of generality, we assume that the victim is z_1 whose MAC address is known to the attacker.

To launch the DeepJam attack, the attacker installs a monitor, which can be a COTS Zigbee device, around the victim device. The monitor sniffs the Zigbee traffic, measures the power level within the Zigbee channel, and also decodes any overheard Zigbee MAC frame. Since the header and FCS of a Zigbee MAC frame are not encrypted, the attacker is aware of the source device of the MAC frame and can also verify whether the frame is corrupted by checking its FCS. If the attacker detects an uncorrupted MAC frame of z_1, it considers the frame being successfully transmitted.

Since the Zigbee traffic is assumed stable within a short period, the attacker attempts to predict the victim's traffic in the near future from the signal overheard in the short past period. Specifically, the attacker uses the deep learning model to predict when z_1 may successfully transmit a MAC frame and then generates a predefined jamming signal, which is fixed Gaussian noise in the targeted Zigbee channel, during the predicted period. The power of the jamming signal is five times larger than that of the Zigbee signal, which is sufficient to disrupt Zigbee transmissions but still much weaker than the WiFi signal's power. So the attacker's action has no significant impact on the WiFi network.

After taking a jamming action, the attacker can determine whether the action is necessary. The signal captured by the adversarial monitor is mixed with the jamming signals, but the attacker can subtract the jamming signal from the sniffed signal to restore the original signal. The jamming action is necessary if the restored signal contains z_1's uncorrupted MAC frame and is unnecessary otherwise.

IV. Problem Formulation

We provide a slotted formulation of the jamming attack and an overview of DeepJam in this section.

A. Slot Duration

We determine the slot duration based on the maximum MAC frame length $\mathcal{L}_z$ of the Zigbee network. In particular, the slot duration $\mathcal{T}_s$ equals $\mathcal{T}_z/2$, where $\mathcal{T}_z$ denotes the time consumed to transmit a Zigbee MAC frame of length $\mathcal{L}_z$.

The choice of the slot duration is critical. First, the slot must be short enough so that the slot status space is of small

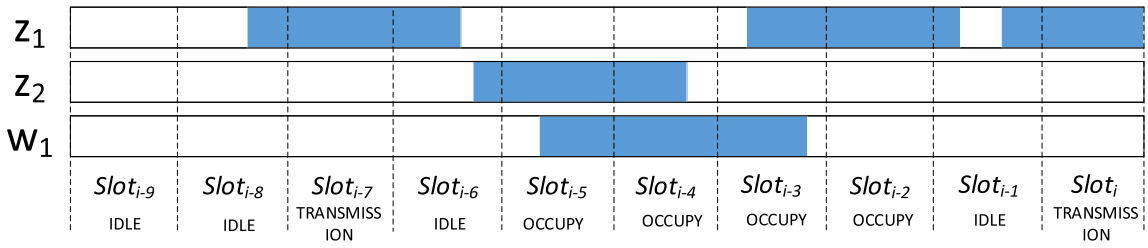


Fig. 3. Slot status.

size. Multiple MAC frames of the victim may appear in one slot, and the slot's status is defined based on the status of those frames (corrupted or not). A proper slot duration must be short so that only a few MAC frames of the victim may be present in one slot. Second, the slot must be long enough so that the optimal action in a future slot is significantly impacted by only a small number of passing slots.

A slot duration of $\mathcal{T}_z/2$ satisfies the need. Since at most two MAC frames of the victim may appear in a slot, the slot has only three possible status which are detailed in Section IV-B. As demonstrated in Section VIII, the number of slots that may have significant impact on the coming slot is at most 10. The evaluation results show that it is feasible to launch DeepJam in real time with a slot duration of $\mathcal{T}_z/2$.

B. Slot Status

We define three slot status: **IDLE**, **TRANSMISSION**, and **OCCUPY**, based on the attacker's observation, and we use an example in Fig. 3 to illustrate how we determine the slot status. Fig. 3 shows an example of the wireless traffic within 10 slots. The device transmits during blue periods and keeps silent during white periods. For simplicity, we assume that only three devices, z_1 (the victim Zigbee device), z_2 (another Zigbee device), and w_1 (a WiFi device), transmit within those 10 slots. The attacker monitors the wireless signal within the Zigbee channel, measures the power level within each slot, and decodes the signal captured in each slot.

In slot_{i-9}, the power level is always below a threshold, and the attacker can conclude that there is no wireless transmission within this slot and considers the slot **IDLE**. In slot_{i-8} and slot_{i-1}, the attacker can detect a MAC frame header containing z_1's address, but the frame does not fully occupy the slot. Those two slots are also considered **IDLE**. If the attacker detects a complete and uncorrupted MAC frame of z_1, the first slot fully occupied by the frame, such as slot_{i-7}, is considered **TRANSMISSION**, and the following slot, such as slot_{i-6}, is considered **IDLE**. Apart from **IDLE** and **TRANSMISSION** slots, all the rest slots are considered **OCCUPY**.

It is worth noting that the attacker may be uncertain about the status of a slot in some cases. For example, the attacker cannot determine the status of slot_i because FCS has not been received yet. In this case, the attacker temporally considers slot_i a **TRANSMISSION** slot.

Three cases can result in an **OCCUPY** slot. First, z_1 does not transmit, but Wi-Fi devices or other Zigbee devices transmit in this slot. slot_{i-5} and slot_{i-4} belong to the first case. Second, z_1 transmits, but Wi-Fi devices transmit simultaneously. Due to interference, the frame of z_1 cannot be correctly decoded. slot_{i-3} belongs to this case. Third, only z_1 transmits in this slot, but the frame's header is within the previous slot and cannot be decoded due to interference. Consequently, the monitor, as well as the destination device, cannot decode the source address correctly to recognize the frame as z_1's. slot_{i-2} belongs to this case.

C. DeepJam Basics

The main idea of DeepJam is predicting the optimal action in a future slot based on the observations in past slots. It takes time to process the captured signal and compute the prediction. Therefore, the attacker cannot immediately obtain the status of a slot and make a proper prediction at the beginning of the next slot. The experimental result in Section VIII shows that the signal processing and prediction computation can be done within one slot. Therefore, we add a gap of one slot between the most recent channel observation and the prediction. To be more specific, within the ith slot, the attacker first processes the signal captured in slot $i-1$. Then the attacker uses the observed channel status before and in slot $i-1$ to predict the optimal action he should take in slot $i+1$.

The attacker can take two actions, **WAIT** or **JAM**, in each slot. If the attacker predicts that the coming slot is a **TRANSMISSION** slot, the attacker takes action **JAM** by sending jamming signals in this slot. Otherwise, the attacker takes action **WAIT** and keeps silent. Since no error correction code is adopted, jamming half of the MAC frame, i.e., one slot, can disrupt the frame with a high probability. Attacker's actions also impact the channel status. If the attacker takes action **JAM** in slot i, the channel status is **OCCUPY** no matter whether there is wireless traffic or not. As we discussed in Section III, the attacker can restore the original signal, so he can determine whether his prediction for an ended slot is correct and thus amend the prediction model.

V. DeepJam Details

DeepJam follows an online-learning process. In particular, the attacker takes actions based on a prediction model and updates the model based on the results of actions. It is well known that Reinforcement Learning (RL) [11] is an effective solution to online-learning problems, so we adopts RL in DeepJam. This section first briefly introduces RL and explains how we convert DeepJam to a typical RL problem. Then we detail the DeepJam RL algorithm.

A. Reinforcement Learning

RL considers an agent who interacts with the environment in a sequence of time slots and tries to maximize some notion of a

cumulative reward [11]. More specifically, the agent observes the state of the environment in the ith slot, denoted by s_i. Based on the observation, the agent takes an action $a_i \in \mathcal{A}_{s_i}$, where $\mathcal{A}_{s_i}$ is the set of possible actions in state s_i. As a result of the action, the agent gets a reward r_{i+1}, and the environment state changes to s_{i+1}. The goal of the agent is to maximize the cumulative reward $R_i \triangleq \sum_{j=i}^{\infty} \gamma^{j-i} r_{j+1}$, where $\gamma \in (0, 1]$ is a discount factor.

To convert DeepJam to a typical RL problem, we define the slot states, the attacker's actions, and the rewards as follows.

1) Slot States: Due to the MAC method, the channel status and the corresponding optimal action of the ith slot are related to the previous slots. We thus abuse the notation and use s_i to represent the state of slot i and refer to the channel status of all the slots that significantly impact the optimal action in slot i. As shown in Section IV, the attacker cannot immediately obtain the status of a slot and complete the prediction at the beginning of the next slot. So s_i does not contain the channel status in slot $i-1$. If we consider N_R slots before slot i except slot $i-1$, s_i can be formulated as a $1 \times N_R$ vector $< u_{i-N_R-1}, \cdots, u_{i-2} >$. Here $u_j \in \{1, 2, 3\}$ indicates the channel status of slot j and is defined as follows:

$$u_j = \begin{cases} 1, & \text{if slot } j \text{ is } \textbf{IDLE}; \\ 2, & \text{if slot } j \text{ is } \textbf{OCCUPY}; \\ 3, & \text{if slot } j \text{ is } \textbf{TRANSMISSION}. \end{cases} \quad (2)$$

A large N_R can achieve a high prediction accuracy but results in long converge time. Since the temporal pattern of the wireless traffic comes from the exponential backoff algorithm, we choose N_R based on the maximum backoff periods of the WiFi and Zigbee networks. Particularly, we calculate N_R as:

$$N_R = \lceil \max(\mathcal{D}_w, \mathcal{D}_z)/\mathcal{T}_s \rceil - 1, \quad (3)$$

where $\mathcal{D}_w$ and $\mathcal{D}_z$ denote the maximum backoff periods in the WiFi and Zigbee networks, respectively; $\mathcal{T}_s$ denotes the duration of a time slot.

2) Actions: We denote the attacker's action in slot i with a_i, where $a_i \in \{\textbf{JAM}, \textbf{WAIT}\}$. The details of **JAM** and **WAIT** actions have been given in Section IV.

3) Rewards: We define the reward of action a_i based on the channel status of slot i. To be more specific, if $a_i =$ **JAM** and slot i is **TRANSMISSION**, the attacker gets a reward $r_{i+1} = R_j \in (0, 1]$ for successfully jamming a MAC frame of the victim. If $a_i =$ **WAIT** and slot i is **IDLE** or **OCCUPY**, the attacker gets a reward $r_{i+1} = R_s \in [0, 1])$ for saving energy. If $a_i =$ **WAIT** and slot i is **TRANSMISSION**, the attacker misses a successfully transmitted MAC frame of the victim and receives a negative reward $r_{i+1} = R_m \in [-1, 0)$. If $a_i =$ **JAM** and slot i is **IDLE** or **OCCUPY**, the attacker wastes energy on an unnecessary jamming and gets a negative reward $r_{i+1} = R_w \in [-1, 0]$.

Attackers can adjust R_j, R_s, R_m, and R_w based on their own constraints. For example, if the energy limitation is a big concern, the attacker can adopt large absolute values for R_w and R_s and adopt a small absolute value for R_m. Since hindering the wireless communication should always be the primary goal of the attacker, R_j and R_m cannot be zero.

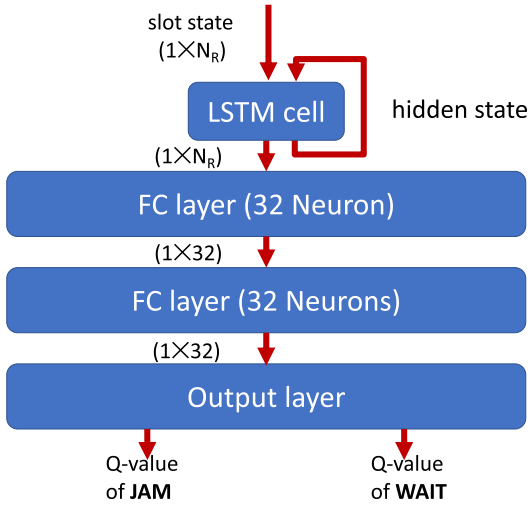


Fig. 4. DeepJam DNN architecture.

B. RL Algorithm of DeepJam

DeepJam adopts Q-Learning (QL) [12], which is a popular RL algorithm, to determine the optimal action in a future slot. QL aims to obtain a Q-function $Q(s, a) \triangleq \mathbb{E}[R_i | s_i = s, a_i = a]$ (also called Q-value) to calculate the expected maximum cumulative reward of taking action a at state s. The action with the maximum Q-value is considered the optimal.

Traditional QL obtains Q-function in a tabular manner which results in long converge time and thus is not suitable for DeepJam. More specifically, the QL algorithm must go through all the combinations of slot states and actions to obtain the Q-function. For example, consider the scenario in Section VIII where the slot state contains the channel status of 10 slots with the slot duration around 2 ms. The QL algorithm takes at least $2 \text{ ms} * 2 * 3^{10} \approx 4$ min to converge and obtain the Q-function. The wireless traffic is highly dynamic, and the temporal pattern is very likely to have changed before the algorithm converges. So DeepJam cannot use traditional QL.

We adopt Deep Q-Learning (DQL) [13] to deal with the large state space and approximate the Q-function with a tailored Deep Neural Network (DNN). The input to the DNN is the slot state, and the outputs are the Q-values of taking **JAM** and **WAIT** actions in the slot. Fig. 4 shows the structure of DeepJam DNN that contains one LSTM cell and two Fully Connected (FC) Layers. We also show the input and output dimensions of each layer in the figure.

The LSTM cell [14] can capture long-time temporal patterns of the wireless traffic. As mentioned in Section V-A.1, the slot state s_i contains the channel status of slots $i - N_R - 1$ to $i - 2$. Since the slots earlier than slot $i - N_R - 1$ may also have non-trivial impact on the optimal action in slot i, we use the LSTM cell to memorize those early slots without increasing the dimension of the slot state. In particular, the LSTM cell contains four gates to maintain a hidden state and calculate the output [14]. The hidden state is affected by the long-time history of inputs and is updated iteratively based on the newly coming input. The four gates control how the hidden state is affected by the newly coming input and how the output is

affected by the hidden state. Therefore, the long-time history of the wireless traffic is memorized by the hidden state of the LSTM cell and contributes to the prediction of optimal actions. The input and output of the LSTM cell are all $1 \times N_R$ vectors. The output of the LSTM cell is fed into two FC layers, both of which contain 32 neurons and adopt the Rectified Liner Unit (ReLU) function as the active functions. The output layer is a linear FC layer that outputs the Q-values of **JAM** and **WAIT** actions.

DeepJam DNN is an estimation of the Q-function, and the training process iteratively reduces the estimation error. The observation on slot i, including the state, the action, and the reward, is one training sample, which is denoted by $o_i =< s_i, a_i, r_{i+1} >$. We adopt the techniques of separate target network and batch gradient descent to stabilize the training process. We maintain a target network whose structure is the same as that of DeepJam DNN. The Q-values given by DeepJam DNN and the target network are denoted by $q(s, a; \Theta_Q)$ and $q'(s, a; \Theta'_Q)$, respectively. Here Θ_Q and Θ'_Q denote the parameter vectors of DeepJam DNN and the target network, respectively; s denotes the slot state; a denotes the action. With o_i as the training sample, the loss function is defined as follows:

$$L_Q(i,\ \Theta_Q) = (r_{i+1} + \gamma \max_a q'(s_{i+1}, a; \Theta'_Q) - q(s_i, a_i; \Theta_Q))^2, \quad (4)$$

where $\gamma \in (0, 1]$ is the discount factor. In each iteration, we update DeepJam DNN's parameter vector with a batch containing five training samples, and the batch is denote by $b =< o_{I(1)}, o_{I(2)}, o_{I(3)}, o_{I(4)}, o_{I(5)} >$, where $I(j)$ is the slot index of the jth training sample. We update Θ_Q as

$$\Theta_Q \leftarrow \Theta_Q - \frac{\rho_Q}{5} \sum_{j=1}^{5} \frac{\partial L_Q(I(j),\ \Theta_Q)}{\partial \Theta_Q} \quad (5)$$

every iteration and replace Θ'_Q with Θ_Q every N_Θ iterations. Here ρ_Q is the learning rate.

We adopt experience replay [15] to accelerate the training process. More specifically, we maintain a memory pool containing the observations on the latest 500 slots. In the training process, we select 20 batches that are continuous in time from the memory pool. To be more specific, if the first batch is $b_1 =< o_{I(1)}, o_{I(2)}, o_{I(3)}, o_{I(4)}, o_{I(5)} >$, the mth batch is $b_m =< o_{I(1)+m-1}, o_{I(2)+m-1}, o_{I(3)+m-1}, o_{I(4)+m-1}, o_{I(5)+m-1} >$. We use the 20 batches to update the DNN parameter vector successively, and the whole process is referred to as one epoch. An attacker can conduct multiple epochs in one slot according to his computational capacity. To capture the long-time temporal pattern, we only initialize the hidden state of the LSTM cell at the beginning of each epoch. We also update the target network at the beginning of each epoch and keep it stable within one epoch. The memory pool is updated at the beginning of each slot. We only take **WAIT** actions in the first $21+N_R$ slots to collect training samples, and we use all the past slots as the memory pool from slot $21 + N_R$ to slot $501 + N_R$. Here N_R is still the aforementioned dimension of the slot state.

In case that DeepJam DNN gets stuck before converging to the optimal estimation of the Q-function, we choose actions in an ϵ-greedy manner. In particular, we take action $a = \arg\max_a q(s_i, a; \Theta_Q)$ in slot i with a probability of $1 - \epsilon$ and take the other action with a probability of ϵ. The hidden state of the LSTM cell is calculated with the states of the 20 slots that are ahead of slot i. The pseudocode of the DeepJam RL algorithm is given in Algorithm 1.

Algorithm 1 RL Algorithm of DeepJam

```
Initialize ϵ, ρ_Q, γ, N_R
Initialize memory pool M
Initialize LSTM hidden state h, DNN parameter vector Θ_Q
Copy Θ_Q to target network parameter vector Θ'_Q
for slot i in DeepJam do
  if i ≤ 21 + N_R then
    take action WAIT
  else
    Calculate LSTM hidden state h_i
    Input s_i to DNN and output q(s_i, a; Θ_Q) for a ∈ {JAM, WAIT}
    Take action according to ϵ-greedy policy
    for each epoch do
      Initialize LSTM hidden state
      Randomly selected 20 continuous batches B
      for each batch in B do
        Θ_Q ← Θ_Q − (ρ_Q/5) Σ_{j=1}^{5} ∂L_Q(I(j), Θ_Q)/∂Θ_Q
      end for
      Θ'_Q ← Θ_Q
    end for
  end if
  Process the signal obtained in slot i − 1
  Update s_{i−1}, a_{i−1}, and r_i to M
end for
```

VI. Countermeasures

We propose two countermeasures against DeepJam for networks with different capabilities.

Networks with advanced devices which have powerful computing capacity can defeat DeepJam by adopting deep learning-based MAC protocols. The victim device can learn the attacker's behavior pattern with DNN and thus predict the slots that are less likely to be jammed. A more powerful victim can train a defense DNN in advance with Generative Adversarial Network (GAN) [16] which can even mislead or manipulate the attacker's behaviors. Specifically, the victim jointly trains two DNNs, including an attack DNN and a defense DNN, in an adversarial manner. Both DNNs take the traffic history as inputs. The attack DNN simulates the attacker and predicts the optimal time to generate the next jamming signal just as DeepJam does. The defense DNN predicts the optimal time to transmit the next MAC frame and attempts to maximize the throughput of the victim. After sufficient rounds of competitions, the defense DNN can capture the

behavior pattern of the attacker, and the victim can partially mitigate the attack's impact by acting according to the defense DNN's prediction. However, most COTS IoT devices only have limited computing capacity and thus cannot adopt the DNN-based countermeasure.

We also propose the dynamic network configuration as a simple yet effective countermeasure against DeepJam. As demonstrated in Section VIII, a sudden change of wireless traffic can immediately and significantly harm the performance of DeepJam, and it takes time for DeepJam DNN to converge again. The victim network can introduce sudden changes to the wireless traffic by changing the network configuration frequently. Consequently, the DeepJam DNN may never converge, and thus the impact of DeepJam is weakened. Our evaluation results show that the dynamic network configuration can indeed harm the performance of DeepJam. However, the dynamic network configuration may bring extra management burden to the network.

VII. Simulating Zigbee and Wi-Fi Networks

Shi *et al.* [17] and Yu *et al.* [18] evaluated the implication of deep learning for wireless security with simulations. Following their work, we built a Matlab simulator to simulate the system in Fig. 1 and evaluated DeepJam with the generated data. We assumed that the WiFi network and the Zigbee network coexist in a 10m $\times$ 10m room, and the locations of the AP, coordinator, WiFi and Zigbee devices were randomly chosen. In the simulation, we set the transmission power of the WiFi device to be 30 times larger than that of the Zigbee device.

A. Simulation of Wi-Fi Network

Zigbee transmissions have negligible impact on WiFi transmissions, so we first simulated the behavior of the WiFi network. Multiple versions of IEEE 802.11 have defined the WiFi standard on 2.4 GHz band. We considered the most popular 802.11g in this paper and leave the analysis of other versions to future work.

We simulated a WiFi network that contains three WiFi devices. The WiFi MAC frame length in the simulator were evenly distributed between 34 bytes (the length of the MAC frame header) and 2,346 bytes (the maximum length of a WiFi MAC frame) with an average of 1,190 bytes. We set the transmission speed of the WiFi network as 54 Mbps, in which case the average transmission time of a MAC frame is around 388 μs (including the time consumed by the acknowledgment). The number of MAC frames generated (not transmitted) by a WiFi device within a short period followed the Poisson distribution illustrated in Section III. In a congested WiFi network, the time consumed to transmit the MAC frames generated during a period of τ is larger than τ, and the number of cached MAC frames increases with time. The number of MAC frames transmitted within one second cannot exceed $\lfloor 1\ \mathrm{s}/388\ \mu\mathrm{s} \rfloor = 2,577$. So we set the arrival rate of a congesting WiFi network as $\lambda_w = 1,000$ fps. We also consider the busy and idle WiFi networks whose arrival rates are set as 500 fps and 100 fps, respectively. The channel is occupied for around 70% and 15% of the time in the busy and idle WiFi networks, respectively. WiFi devices accessed the channel with CSMA/CA, and the maximum backoff period was 20,460 μs as defined in the standard. In the PHY layer of the WiFi network, we adopted the Additive White Gaussian Noise (AWGN) model to simulate the WiFi channel. We set the SNR as 14 dB, roughly the power ratio of the WiFi signal to the Zigbee signal.

B. Simulation of Zigbee Network

We simulated the behavior of the Zigbee network after obtaining the WiFi traffic. We considered a congesting Zigbee network in which all the MAC frames were of the maximum length, 127 bytes. It takes 4,256 μs to transmit one MAC frame. The generation of MAC frames in the Zigbee device was also a Poisson process. For similar reasons explained in Section VII-A, we set the arrival rate of the busy Zigbee network as λ_z =260 fps, in which case a single Zigbee device can fully occupy the channel. We simulated both beacon-enabled and beacon-disabled Zigbee networks. The superframe in the beacon-enabled network only contained the active period, and the length of the superframe was 15,360 μs. Zigbee devices accessed the channel with slotted CSMA/CA in the beacon-enabled network and with unslotted CSMA/CA in the beacon-disabled network. The maximum backoff periods of beacon-enabled and beacon-disabled Zigbee networks were both 9,920 μs as defined in IEEE 802.15.4. We also simulated both Energy Detection (ED) or Carrier Sensing (CS) for CCA. In the simulation with ED, Zigbee devices could detect both Zigbee and Wi-Fi signals and considered the channel idle if and only if there was no wireless traffic. In the simulation with CS, Zigbee devices cannot detect WiFi signals and considered the channel idle if it did not detect Zigbee signals. Since our study focuses on the efficiency of the jamming strategy, i.e., the ratio of the jammed MAC frames, we considered a Zigbee network without acknowledgment for the convenience of analysis.

We used the MATLAB Communications Toolbox Library for Zigbee [19] to simulate the Zigbee PHY layer. More specifically, we used the AWGN model to simulate the Zigbee channel. We assumed that the noise in the wireless environment is negligible, and the Zigbee PHY frame can always be successfully transmitted without the interference from other wireless signals. If another wireless signal was present during the transmission of a Zigbee PHY frame, we considered that wireless signal white Gaussian noise. Finally, we obtained the number of incorrectly decoded bits in a Zigbee MAC frame from the Matlab simulator. Since the Cyclic Redundancy Check (CRC) code adopted by the Zigbee network can only correct one incorrectly decoded bit in a MAC frame, we considered the transmission of a MAC frame failed if more than one bit were incorrectly decoded.

VIII. Evaluation

This section first introduces the evaluation setup, including the hardware and software, evaluation metrics, parameter setting, and comparison method. Then we evaluate our scheme

in various scenarios. Since Zigbee networks in different application scenarios may differ in MAC and CCA methods, we evaluate DeepJam's performance under different Zigbee network configurations. We also evaluate the impacts of the WiFi network's traffic load and the Zigbee network size. Finally, we evaluate the latency of DeepJam and the efficiency of the countermeasure.

A. Evaluation Setup

1) Hardware and Software: We implemented DeepJam DNN with PyTorch 1.4 [20]. To make minimum assumptions about the attacker's ability, we conducted all the experiments on a COTS personal computer. The computer is equipped with an Intel Core i7-3770 3.4 GHz CPU, where all the computations were conducted.

2) Evaluation Metrics: We used two metrics to evaluate DeepJam. The first is **hit rate** (HR) defined as

$$\text{HR} = \mathcal{N}_h/\mathcal{N}_j, \tag{6}$$

where $\mathcal{N}_j$ denotes the number of slots that are predicted to be **TRANSMISSION** and thus jammed by the attacker; $\mathcal{N}_h$ denotes the number of the victim's MAC frames that are corrupted not by the wireless signal of other devices but by the jamming signal. In other words, $\mathcal{N}_h$ predictions among the $\mathcal{N}_j$ **TRANSMISSION** predictions are correct, so HR reflects the prediction accuracy of the DeepJam DNN.

We define the second metric **jam rate** (JR) as

$$\text{JR} = \mathcal{N}_h/(\mathcal{N}_h + \mathcal{N}_s), \tag{7}$$

where $\mathcal{N}_s$ denotes the number of the victim's MAC frames that are successfully transmitted.

HR measures the ratio of necessary jamming actions, and JR measures the victim's throughput decrease due to jamming. The jamming attack is said to be efficient if both HR and JR are high; i.e., the attacker can significantly reduce the victim's throughput with limited energy consumption.

3) Parameter Setting: We adopted 0.8, 0.1, -0.8, and -0.3 as R_j, R_s, R_m, and R_w, respectively. The discount factor and the learning rate are 0.9 and 0.01, respectively. We trained the DNN for five epochs in each slot. The slot duration $\mathcal{T}_s$ equals 2,128 μs, and slot state's dimension N_R equals 10.

4) Comparison Methods: We compare DeepJam with conventional random jamming and reactive jamming as follows. We first launch DeepJam for 10,000 slots among which $\mathcal{N}_j^d$ slots are jammed and calculate the HR and JR, denoted by HR_d and JR_d. Then we launch random jamming and reactive jamming each for 10,000 slots, aiming to achieve the same jam rate as DeepJam's. To be more specific, we jam each slot with a probability of JR_d in random jamming and jam the slot that follows a slot containing a Zigbee preamble with a probability of JR_d in reactive jamming. The number of jamming actions token in random jamming and reactive jamming are denoted by $\mathcal{N}_j^{ra}$ and $\mathcal{N}_j^{re}$, respectively. We also calculate the hit rates of random jamming and reactive jamming, denoted by HR_{ra} and HR_{re}, respectively. We define a metric **jamming-efficiency gain** (JEG) for more straightforward comparison. The JEGs of DeepJam over random jamming and reactive jamming are defined as $\text{JEG}_{d/ra} = \mathcal{N}_j^{ra}/\mathcal{N}_j^d$ and $\text{JEG}_{d/re} = \mathcal{N}_j^{re}/\mathcal{N}_j^d$, respectively. JEG measures the advantage of DeepJam over random jamming and reactive jamming in terms of the jamming actions' efficiency.

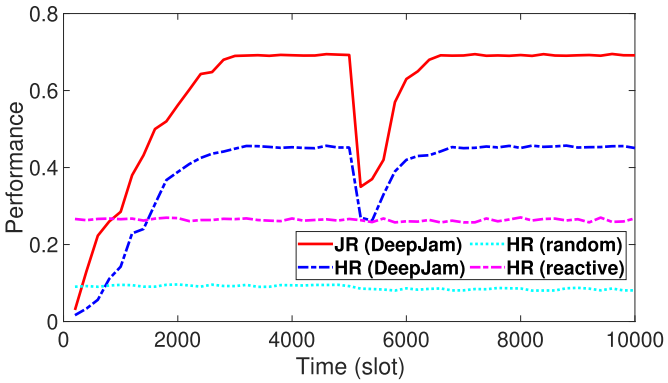


Fig. 5. Comparison of DeepJam, random jamming, and reactive jamming.

B. Efficiency of DeepJam

In this subsection, we first compare DeepJam, random jamming, and reactive jamming in a specific scenario. Then we evaluate the impact of multiple parameters.

1) Performance Comparison: The initial configuration of the experiment was as follows. The arrival rate of the WiFi network was $\lambda_w = 500$ fps. The Zigbee network contained three Zigbee devices, was beacon-disabled, and adopted CS for CCA. We changed the configuration at slot 5,000 by adjusting λ_w to 300 fps. The HRs and JRs calculated for every 200 slots are shown in Fig. 5.

With the initial configuration, DeepJam converged after around 3,100 slots (about 6.3 sec) and achieved an HR_d of 0.43 and a JR_d of 0.68. The performance of DeepJam dramatically decreased around slot 5,000 due to the sudden change of wireless traffic, but DeepJam converged again after about 3.3 sec and achieved an HR_d of 0.44 and a JR_d of 0.69. With short converge time, DeepJam can handle the highly dynamic wireless traffic. DeepJam significantly outperformed random jamming and reactive jamming with an $\text{JEG}_{d/ra}$ of 4.9 and an $\text{JEG}_{d/re}$ of 1.7.

To better understand the comparison between DeepJam and reactive jamming, it is worth noting that reactive jamming is triggered by the detection of Zigbee preambles. So reactive jamming wastes energy on jamming Zigbee traffic which is not from the victim device. Besides, some corrupted Zigbee packets contain correct preambles which also trigger a reactive jammer to launch unnecessary jamming. Therefore, DeepJam improved the HR of reactive jamming by more than 1.7 times because it can identify the uncorrupted packets of the victim more accurately.

2) Impact of WiFi Traffic Loads: In the experiment, the Zigbee network contained three Zigbee devices, was beacon-disabled, and adopted CS for CCA. We first evaluated DeepJam with a congesting WiFi network ($\lambda_w = 1,000$ fps) and then repeated the evaluation with the busy and idle WiFi networks ($\lambda_w = 500$ fps and $\lambda_w = 100$ fps, respectively).

TABLE I
PERFORMANCE WITH DIFFERENT WIFI TRAFFIC LOADS

λ_w	1,000 fps	500 fps	100 fps
HR_d	0.12	0.43	0.45
JR_d	0.30	0.67	0.67
converge time	15.5 sec	6.9 sec	6.7 sec
HR_{ra}	0.03	0.07	0.09
HR_{re}	0.14	0.23	0.26
$JEG_{d/re}$	4.0	6.1	5.0
$JEG_{d/re}$	0.86	1.8	1.7

TABLE II
PERFORMANCE WITH DIFFERENT MAC AND CCA METHODS

	scenario 1	scenario 2	scenario 3	scenario 4
HR_d	0.37	0.40	0.45	0.44
JR_d	0.65	0.64	0.72	0.72
converge time	7.6 sec	7.1 sec	6.7 sec	6.7 sec
HR_{ra}	0.05	0.05	0.08	0.07
HR_{re}	0.23	0.23	0.25	0.24
$JEG_{d/ra}$	7.4	8.0	5.6	6.2
$JEG_{d/re}$	1.6	1.7	1.8	1.8

For each evaluation, we launched DeepJam for 50 times. The average converge time and the average HR_d and JR_d after convergence are shown in Table I. The table also lists HR_{ra}, HR_{re}, $JEG_{d/ra}$, and $JEG_{d/re}$ for comparison.

DeepJam performs well with the busy and idle WiFi networks, but its performance in a congested WiFi network is less satisfactory. We found that 95% of the victim's frames were corrupted due to the interference of the congested WiFi network. So the training samples, i.e., slots, containing uncorrupted transmissions were few, which resulted in long converge time and poor performance. However, the throughput of the victim is extremely low when it coexists with a congested WiFi network, so it does not make sense for the attacker to launch the jamming attack in such a scenario. DeepJam performs similarly with busy and idle WiFi networks. So we only considered the busy WiFi network hereafter, which is more common in practice.

3) Impact of MAC and CCA Methods: The Zigbee network in this experience contained three Zigbee devices, and the WiFi network was busy. We considered four scenarios: a beacon-enabled Zigbee network adopting CS (scenario 1), a beacon-enabled Zigbee network adopting ED (scenario 2), a beacon-disabled Zigbee network adopting CS (scenario 3), and a beacon-disabled Zigbee network adopting ED (scenario 4). We repeated DeepJam 50 times in each scenario and calculated the average converge time, HRs, and JRs. The results are shown in Table II. The table also lists the corresponding HR_{ra}, HR_{re}, $JEG_{d/ra}$, and $JEG_{d/re}$ for comparison.

In all scenarios, DeepJam significantly outperformed random jamming and reactive jamming in terms of energy efficiency and converged within 8 sec. CCA methods had no significant impact on DeepJam, while the performance of DeepJam with beacon-disabled Zigbee networks was slightly better than that with beacon-enabled ones. The main reason is that slotted CSMA/CA is more complex than unslotted CSMA/CA. In particular, the end of the backoff period is synchronized with slot boundaries, and the device may conduct CCA multiple times before transmission even though the channel is idle, resulting in more chaotic wireless traffic. So DeepJam converged slower with beacon-enabled Zigbee networks, and HR_d and JR_d were both lower as well. To evaluate DeepJam in the worst case, we considered beacon-enabled Zigbee networks adopting CS hereafter.

TABLE III
PERFORMANCE WITH DIFFERENT NUMBERS OF ZIGBEE DEVICES

$\mathcal{N}_D$	1	2	3	4	5
HR_d	0.39	0.41	0.37	0.39	0.37
JR_d	0.62	0.65	0.65	0.62	0.60
Converge time	6.3 sec	6.3 sec	6.7 sec	6.7 sec	6.3 sec
HR_{ra}	0.23	0.19	0.08	0.06	0.05
HR_{re}	0.71	0.32	0.24	0.18	0.13
$JEG_{d/ra}$	1.7	2.2	5.6	6.5	7.4
$JEG_{d/re}$	0.55	1.3	1.8	2.2	2.8

4) Impact of Zigbee Devices' Amount $\mathcal{N}_D$: This experiment considered a beacon-enabled Zigbee network adopting CS and an busy WiFi network. We evaluated $\mathcal{N}_D$ equal to 1, 2, 3, 4, or 5. For each value, we launched DeepJam 50 times. The average HRs, JRs, and converge time are shown in Table III. The table also lists the corresponding HR_{ra}, HR_{re}, $JEG_{d/ra}$, and $JEG_{d/re}$ for comparison.

DeepJam outperformed random jamming in all scenarios, and the advantage of DeepJam over reactive jamming became more significant with the increase of $\mathcal{N}_D$ (especially for $\mathcal{N}_D \geq 3$). DeepJam achieved an $JEG_{d/re}$ of 2.8 with $\mathcal{N}_D$ equaled 5. With the increase of $\mathcal{N}_D$, reactive jamming wasted more energy on jamming Zigbee traffic that was not from the victim, thus decreasing the HR_{re}. In contrast, the impact of $\mathcal{N}_D$ on DeepJam was negligible. HR_d, JR_d, and the converge time of DeepJam were stable for different $\mathcal{N}_D$.

C. Latency

This section evaluated the latency of DeepJam. To launch DeepJam in real time, the attacker needs to process the data captured in the previous slot, calculate the output of the deep neural network, and update the deep neural network within one slot, i.e., 2,128 μs. We denote the time taken to process the captured data, calculate the DNN output, and update the DNN by T_p, T_c, and T_u, respectively. As mentioned in Section V, the attacker can finish the three tasks in parallel, so DeepJam is feasible in real time if $\max(T_p,\ T_c,\ T_u) \leq 2,128\ \mu s$. As defined in 802.15.4, the interval between two continuous MAC frames is 40 symbols during which the Zigbee device can process a MAC frame. We assume that the attacker uses a COTS Zigbee receiver as the monitor, so T_p is no more than 40 symbols, i.e., 640 μs. In all the aforementioned experiments, the maximum values of T_c and T_u are 10 μs and 1,870 μs, respectively. Both are less than 2,128 μs. Therefore, the real-time DeepJam attack is feasible.

D. Efficiency of Dynamic Network Configuration

Finally, we evaluated the efficacy of the dynamic network configuration countermeasure. This section reports the results for two configurations, the backoff unit and the superframe

length. We considered a beacon-enable Zigbee network that adopts CS and contained 3 Zigbee devices in this experiment.

We first evaluated the efficacy of dynamic backoff units. The backoff units of the Zigbee network were chosen from {20, 40, 60, 80, 100} samples, and the network randomly changed the backoff unit every 2 sec. We launched DeepJam in this scenario for 20 min. The average HR_d and JR_d were only 0.21 and 0.37, respectively; and the $\text{JEG}_{d/ra}$ and $\text{JEG}_{d/re}$ were 2.3 and 0.78, respectively. Then we evaluated the dynamic superframe length, which was randomly chosen from {10,240, 15,360, 20,480, 25,600} μs every 2 sec. We also launched DeepJam in this scenario for 20 min. The average HR_d and JR_d were only 0.16 and 0.23, respectively; and $\text{JEG}_{d/ra}$ and $\text{JEG}_{d/re}$ were 1.7 and 0.59, respectively.

The performance of DeepJam with dynamic network configurations was much worse than that with stable network configurations. Due to the network-configuration changes, the wireless traffic pattern always changed before DeepJam DNN converged, which effectively weakened DeepJam's impact. The evaluation results show that dynamic network configuration is indeed an effective countermeasure against DeepJam.

IX. Related Work

With the surge of IoT devices, the cross-technology interference has become a critical problem. There have been many studies evaluating the impact of the cross-technology interference [9], [10] or exploring new media access control schemes to mitigate it [18], [21]. In this paper, we focus on another negative consequence of the cross-technology interference, which makes the wireless network more vulnerable to the jamming attack.

There has been significant research on jamming attacks and defenses. Traditional jammers can be classified into constant jammers, deceptive jammers, random jammers, and reactive jammers [22], [23]. A reactive jammer only disrupts the targeted channel upon detection of a specific signal such as the Zigbee MAC frame header, so it significantly outperforms other categories of jammers in terms of efficiency. In spite of the strict real-time requirement, researchers have successfully implemented the reactive jammer in the real environment [2]. However, the reactive jammer becomes less efficient in the CTI context. Specifically, the detection-based jamming strategy would waste energy on jamming the victim's transmissions which have been disrupted by CTI and also the transmissions of other devices.

With the surge of deep learning, there have been recent studies adopting deep neural networks to launch or mitigate the jamming attack [17], [24], [25]. The work in [17] is the most related to ours. Shi *et al.* [17] considered a slotted cognitive network in which the victim accesses the channel with a deep learning-based media access control method. They proposed a generative adversarial network-based jamming strategy which is efficient as shown in their experimental result. Unlike the scenario considered in [17], we consider a more realistic, chaotic scenario with cross-technology interference, in which an unslotted Wi-Fi network coexists with an unslotted Zigbee network.

X. Conclusion

In this paper, we present the design of DeepJam, a deep learning-guided jamming strategy which exploits CTI in complex wireless environments. Detailed evaluations confirm that DeepJam is more stealthy and energy-efficient than conventional random and reactive jamming strategies. We also propose a simple yet effective countermeasure against DeepJam.

Acknowledgment

The authors truly appreciate the anonymous reviewers for their constructive comments.

References

[1] Y. Chen, W. Xu, W. Trappe, and Y. Zhang, *A Brief Survey of Jamming and Defense Strategies*. Boston, MA, USA: Springer, 2009.

[2] M. Wilhelm, I. Martinovic, J. Schmitt, and V. Lenders, "Short paper: Reactive jamming in wireless networks: How realistic is the threat?" in *Proc. WiSec*, Hamburg, Germany, Jun. 2011, pp. 47–52.

[3] S. Zhao, Z. Lu, Z. Luo, and Y. Liu, "Orthogonality-sabotaging attacks against OFDMA-based wireless networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Paris, France, Apr. 2019, pp. 1603–1611.

[4] Q. Yan, H. Zeng, T. Jiang, M. Li, W. Lou, and Y. T. Hou, "Jamming resilient communication using MIMO interference cancellation," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 7, pp. 1486–1499, Jul. 2016.

[5] S. Chen, K. Zeng, and P. Mohapatra, "Jamming-resistant communication: Channel surfing without negotiation," in *Proc. IEEE Int. Conf. Commun.*, May 2010, pp. 1–6.

[6] X. Zhang, P. Huang, L. Guo, and Y. Fang, "Hide and seek: Waveform emulation attack and defense in cross-technology communication," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Dallas, UT, USA, Jul. 2019, pp. 1117–1126.

[7] Y. Shi, K. Davaslioglu, and Y. E. Sagduyu, "Generative adversarial network for wireless signal spoofing," in *Proc. ACM Workshop Wireless Secur. Mach. Learn. (WiseML)*, 2019, pp. 55–60.

[8] Y. Shi, T. Erpek, Y. E. Sagduyu, and J. H. Li, "Spectrum data poisoning with adversarial deep learning," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 407–412.

[9] R. G. Garroppo, L. Gazzarrini, S. Giordano, and L. Tavanti, "Experimental assessment of the coexistence of Wi-Fi, ZigBee, and Bluetooth devices," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2011, pp. 1–9.

[10] R. Musaloiu-E and A. Terzis, "Minimising the effect of WiFi interference in 802.15. 4 wireless sensor networks," *Int. J. Sensor Netw.*, vol. 3, no. 1, pp. 43–54, 2008.

[11] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[12] C. Watkins, "Learning from delayed rewards," King's College, Cambridge, U.K., Tech. Rep., 1989.

[13] I. Goodfellow, Y. Bengio, and A. Courville, Eds., *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[16] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[17] Y. Shi, Y. E. Sagduyu, T. Erpek, K. Davaslioglu, Z. Lu, and J. H. Li, "Adversarial deep learning for cognitive radio security: Jamming attack and defense strategies," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–6.

[18] Y. Yu, T. Wang, and S. C. Liew, "Deep-reinforcement learning multiple access for heterogeneous wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1277–1290, Jun. 2019.

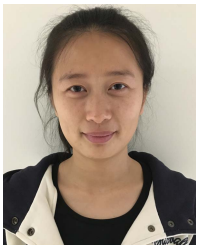
[19] *Communications Toolbox Library for the Zigbee Protocol*. Accessed: May 22, 2021. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/62845-communications-toolbox-library-for-the-zigbee-protocol

[20] *Pytorch*. Accessed: May 22, 2021. [Online]. Available: https://pytorch.org/

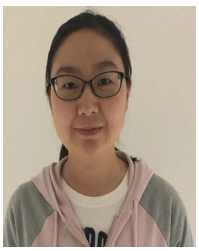
[21] X. Zhang and K. Shin, "Enabling coexistence of heterogeneous wireless systems: Case for ZigBee and WiFi," in *Proc. 12th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2011, pp. 1–11.
[22] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy, "Denial of service attacks in wireless networks: The case of jammers," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 2, pp. 245–257, 2nd Quart., 2011.
[23] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *Proc. 6th ACM Int. Symp. Mobile Ad Hoc Netw. Comput. (MobiHoc)*, Urbana-Champaign, IL, USA, 2005.
[24] F. Slimeni, B. Scheers, Z. Chtourou, and V. Le Nir, "Jamming mitigation in cognitive radio networks using a modified Q-learning algorithm," in *Proc. Int. Conf. Mil. Commun. Inf. Syst. (ICMCIS)*, May 2015, pp. 1–7.
[25] S. Machuzak and S. K. Jayaweera, "Reinforcement learning based anti-jamming with wideband autonomous cognitive radios," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Jul. 2016, pp. 1–5.

Dianqi Han (Graduate Student Member, IEEE) received the B.S. degree in information security from the University of Science and Technology of China and the M.E. degree in computer and electrical engineering from the University of California, Davis. He is currently pursuing the Ph.D. degree in computer engineering with Arizona State University. His research interests include indoor navigation, security and privacy issues in mobile systems, and machine learning in wireless networks.

Ang Li received the B.E. degree in network engineering from Guangxi University, China, in 2010, and the M.S. degree in computer science from Beihang University, China, in 2014. He is currently pursuing the Ph.D. degree in computer engineering with Arizona State University. His research interest is about security and privacy in social networks, machine learning, wireless networks, and mobile computing.

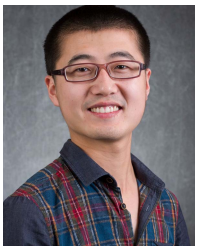


Lili Zhang received the B.S. degree in information security from the University of Science and Technology of China in 2016. She is currently pursuing the Ph.D. degree in computer engineering with Arizona State University. Her research interest is about cyber security and privacy issues in mobile systems.

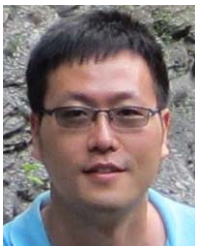


Yan Zhang (Graduate Student Member, IEEE) received the B.S. degree in information and computing science from Xi'an Jiaotong University, China, in 2014, and the M.S. degree in communication and information system from Beijing Normal University in 2017. She is currently pursuing the Ph.D. degree in computer engineering with Arizona State University. Her research interest is about cyber security and privacy issues in mobile systems.

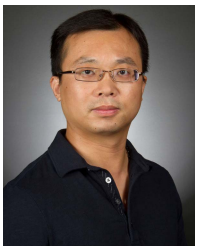
Jiawei Li (Graduate Student Member, IEEE) received the B.E. degree in telecommunication engineering from the Nanjing University of Posts and Telecommunications in 2013. He is currently pursuing the Ph.D. degree in computer engineering with Arizona State University. His research interest is on security and privacy issues in wireless network and wireless sensing.



Tao Li (Member, IEEE) received the B.E. degree in software engineering from Hangzhou Dianzi University in 2012, the M.S. degree in computer science and technology from Xi'an Jiaotong University in 2015, and the Ph.D. degree in computer engineering from Arizona State University in 2020. He is currently an Assistant Professor with the Department of Computer and Information Technology, Indiana University-Purdue University Indianapolis (IUPUI). His primary research is on security and privacy issues in networked/mobile/distributed systems, smart sensing, and wireless networks.



Ting Zhu (Senior Member, IEEE) received the Ph.D. degree from the University of Minnesota, Twin City, in 2010. He is currently an Associate Professor with the Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore. He has authored or coauthored over 120 articles in premier journals and over 120 papers in premier conferences. His research interests include cyber-physical systems, intelligent building systems, the IoT, wireless networks, and mobile systems, which are supported by the NSF, Microsoft, and other agencies. He has served on many program committees of premier conferences. He was the recipient of the NSF CAREER Award in 2017 and CRA Computing Innovation Fellowship in 2010. He currently serves as an editorial board member for three international journals. He has received a number of research awards in the areas of energy efficient smart buildings, networking, and the Internet of Things (IoT).



Yanchao Zhang (Fellow, IEEE) received the B.E. degree in computer science and technology from the Nanjing University of Posts and Telecommunications in 1999, the M.E. degree in computer science and technology from the Beijing University of Posts and Telecommunications in 2002, and the Ph.D. degree in electrical and computer engineering from the University of Florida in 2006. He is currently a Professor with the School of Electrical, Computer and Energy Engineering, Arizona State University. His primary research interests are network and distributed system security, wireless networking, and mobile computing. He is an IEEE Fellow for contributions to wireless and mobile security. He received the U.S. NSF CAREER Award in 2009. He chaired the 2017 IEEE Conference on Communications and Network Security (CNS), the 2016 ARO-funded Workshop on Trustworthy Human-Centric Social Networking, the 2015 NSF Workshop on Wireless Security, and the 2010 IEEE GLOBECOM Communication and Information System Security Symposium. He is/was on the editorial boards of IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON CONTROL OF NETWORK SYSTEMS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.