Cracking Channel Hopping Sequences and Graph Routes in Industrial TSCH Networks

XIA CHENG, JUNYANG SHI, and MO SHA*, State University of New York at Binghamton, USA

Industrial networks typically connect hundreds or thousands of sensors and actuators in industrial facilities, such as manufacturing plants, steel mills, and oil refineries. Although the typical industrial Internet of Things (IoT) applications operate at low data rates, they pose unique challenges because of their critical demands for reliable and real-time communication in harsh industrial environments. IEEE 802.15.4-based wireless sensor-actuator networks (WSANs) technology is appealing for use to construct industrial networks because it does not require wired infrastructure and can be manufactured inexpensively. Battery-powered wireless modules easily and inexpensively retrofit existing sensors and actuators in industrial facilities without running cables for communication and power. To address the stringent real-time and reliability requirements, WSANs made a set of unique design choices such as employing the Time-Synchronized Channel Hopping (TSCH) technology. These designs distinguish WSANs from traditional wireless sensor networks (WSNs) that require only best effort services. The function-based channel hopping used in TSCH simplifies the network operations at the cost of security. Our study shows that an attacker can reverse engineer the channel hopping sequences and graph routes by silently observing the transmission activities and put the network in danger of selective jamming attacks. The cracked knowledge on the channel hopping sequences and graph routes is an important prerequisite for launching selective jamming attacks to TSCH networks. To our knowledge, this paper represents the first systematic study that investigates the security vulnerability of TSCH channel hopping and graph routing under realistic settings. In this paper, we demonstrate the cracking process, present two case studies using publicly accessible implementations (developed for Orchestra and WirelessHART), and provide a set of insights.

CCS Concepts: • **Security and privacy → Mobile and wireless security**; • **Networks → Link-layer protocols**.

Additional Key Words and Phrases: Time-Synchronized Channel Hopping, Graph Routing, Selective Jamming Attack, IEEE 802.15.4e, Industrial Wireless Sensor-Actuator Networks

ACM Reference Format:
Xia Cheng, Junyang Shi, and Mo Sha. 2020. Cracking Channel Hopping Sequences and Graph Routes in Industrial TSCH Networks. *ACM Trans. Internet Technol.* 1, 1, Article 1 (May 2020), 28 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

The Internet of Things (IoT) refers to a broad vision whereby things, such as everyday objects, places, and environments, are interconnected with one another via the Internet [36]. Until recently, most of the IoT infrastructures and applications developed by businesses have focused on smart

*Corresponding author

Part of this article was published in Proceedings of the IoTDI [7].
Authors' address: Xia Cheng; Junyang Shi; Mo Sha, State University of New York at Binghamton, 4400 Vestal Parkway East, Binghamton, NY, 13902, USA, {xcheng12,jshi28,msha}@binghamton.edu.

1533-5399/2020/5-ART1 $15.00
https://doi.org/0000001.0000001

homes and wearables. However, it is the "production and manufacturing" IoT, which underlies the Fourth Industrial Revolution (or Industry 4.0), that promises to be one of the largest potential economic effects of IoT [48] — up to $47 trillion in added value globally by 2025, according to McKinsey's report on future disruptive technologies [28].

Industrial networks, the underlying support of industrial IoT, typically connect hundreds or thousands of sensors and actuators in industrial facilities, such as manufacturing plants, steel mills, oil refineries, and infrastructures implementing complex monitoring and control processes. Although the typical industrial applications operate at low data rates, they pose unique challenges because of their critical demands for *reliable* and *real-time* communication in harsh industrial environments. Failure to achieve such performance can lead to production inefficiency, safety threats, and financial loss. These requirements have been traditionally met by specifically chosen wired solutions, e.g., the Highway Addressable Remote Transducer (HART) communication protocol [17], where cables connect sensors and forward sensor readings to a control room where a controller collects sensor readings and sends commands to actuators. However, wired networks are often costly to deploy and maintain in industrial environments and difficult to reconfigure to accommodate new production requirements. IEEE 802.15.4-based wireless sensor-actuator networks (WSANs) technology is appealing for use in industrial applications because it does not require wired infrastructure and can be manufactured inexpensively. Battery-powered wireless modules easily and inexpensively retrofit existing sensors and actuators in industrial facilities without running cables for communication and power.

There have been two major technology breakthroughs in industrial WSANs. An initial breakthrough came in 1997 from the UC Berkeley's Smart Dust project [33], which demonstrated tiny, low-power motes could sense, compute, and communicate through wireless mesh networks. A second breakthrough came in 2006 with the time-synchronized mesh protocol (TSMP) [34] with a core technology of Time-Synchronized Channel Hopping (TSCH): All devices in a network are time synchronized and hop channels to exploit frequency diversity. The TSCH technology was adopted by the leading industrial WSAN standards (WirelessHART [54] and ISA100 [19]) and the one being standardized by IETF (6TiSCH [18]). A decade of real-world deployments of WirelessHART and ISA100 have demonstrated the feasibility of using TSCH-based WSANs to achieve reliable low-power wireless communication in industrial facilities. Therefore, TSCH was amended into the IEEE 802.15.4e standard in 2012 [1] as a mode to support industrial or embedded applications with critical performance requirements.

To address the stringent real-time and reliability requirements, TSCH made a set of unique design choices. These designs distinguish TSCH from traditional Medium Access Control (MAC) protocols designed for the wireless sensor networks (WSNs) that require only best effort services [26]. Specifically, TSCH divides time into slices of fixed length that are grouped in a slotframe. Nodes are synchronized and share the notion of a slotframe that repeats over time. Frequency diversity is used to mitigate effects of multipath fading and to improve the robustness and the network capacity. Channel hopping is achieved by sending successive packets on different frequencies. All devices in the network compute the channel hopping sequences by following a function. Reliable graph routing is used to enhance the network reliability by taking advantage of route diversity and redundancy. For each data flow, the graph routing provides a primary routing path and multiple backup routes. TSCH's function-based channel hopping simplifies the network operations at the cost of security. Our study shows that an attacker can reverse engineer the channel hopping sequences by silently observing the channel activities and put the network in danger of selective jamming attacks, where the attacker jams only the transmission of interest on its specific communication channel in its specific time slot, which makes the attacks energy efficient and hardly detectable. The selective jamming attacks are more severe threats in WSANs compared to the simple jamming attacks,

Table 1. Comparison of Jamming Effect

Jamming Type	PRR w/o Jamming	PRR w/ Jamming	Packets	Energy Consumption
Constant	0.955	0.338	571428	188091.24 mJ
Random	0.963	0.784	133333	45935.89 mJ
Selective	0.942	0.1615	2000	3201.24 mJ

because jamming a channel or the whole spectrum continuously can be easily detected and located by a wireless intrusion prevention system (WIPS) [39, 55, 56, 59]. Many countermeasures have been developed in the WSN literature to minimize the damage, such as adjusting routing [10, 20, 57]. However, the existing approaches may fail to detect more stealthy attacks such as selective jamming attacks, because the transmission failures caused by the attacks only happen occasionally and are buried in the normal fluctuations of low-power links. We perform three sets of experiments to compare the effect of different types of jamming attacks: constant jamming, random jamming, and selective jamming. We let the sender transmit 2,000 data packets to the receiver following a pre-computed transmission schedule with a length of 1,950*s* and configure a jammer to generate jamming signals using different attacking methods. As Table 1 shows, selective jamming introduces most significant damage to the packet receptions at the cost of the least amount of jamming packets and energy. To launch selective jamming to a TSCH-based WSAN, the attacker not only needs to crack the channel hopping sequences, but also needs to derive the routing paths in the network. Our study shows that the attacker can crack the routes used by the graph routing in WirelessHART networks by silently observing the packet transmission activities.

To our knowledge, this paper represents the first systematic study that investigates the security vulnerability of TSCH channel hopping and graph routing under realistic settings. The contributions of this work are four-fold:

- We present the security vulnerability of TSCH channel hopping in IEEE 802.15.4e by demonstrating the process of cracking the channel hopping sequences;
- We show the security vulnerability of graph routing in WirelessHART networks by demonstrating the cracking process;
- We perform two case studies using publicly accessible implementations[1];
- We provide a set of insights distilled from our analysis and case studies to secure the network by increasing the cracking difficulty.

The remainder of the paper is organized as follows. Section 2 introduces the background of TSCH channel hopping and graph routing. Section 3 presents the security vulnerability by demonstrating the cracking process. Section 4 analyzes the security vulnerability of graph routing in WirelessHART networks. Section 5 and Section 6 describe two case studies. Section 7 presents our lessons learned. Section 8 reviews related work. Section 9 concludes the paper.

2 BACKGROUND ON TSCH CHANNEL HOPPING AND GRAPH ROUING

To provide time-deterministic packet deliveries and combat narrow-band interference and multi-path fading, TSCH combines time-slotted MAC access, multi-channel communication, and channel hopping. TSCH divides time into slices of fixed length that are grouped in a slotframe. Each time slot is long enough to deliver a data packet and an acknowledgement between a pair of devices. Nodes are synchronized and share the notion of a slotframe that repeats over time. Channel hopping is achieved by sending successive packets on different frequencies. TSCH uses the channel hopping

[1]To avoid bias in our attack design and experiments, we use the implementations provided by the third party researchers in our case studies and have different authors to design the attacking program and configure the networks to collect data traces.

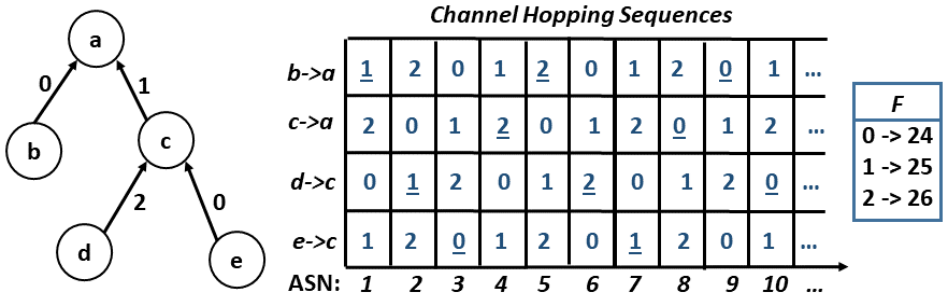


Fig. 1. Example channel hopping sequences with three channels used. The number beside each link indicates the $ChannelOffset$ assigned to that link.

sequences, which are fixed and known by all devices in the network, instead of using the pseudo-random ones to minimize the channel synchronization overhead. Initially, 16 different channels are available for communication. Each channel is identified by $ChannelOffset$. However, some of these frequencies could be blacklisted due to low-quality communication and, hence, the total number of channels $N_{channel}$ used for channel hopping may be fewer than 16. In TSCH, a link is defined as the pairwise assignment of directed communication between two nodes in a given time slot on a given channel offset. Hence, a link between two communicating nodes can be represented by a pair of values that specifies the time slot in the slotframe and the channel offset used by the nodes in that time slot. Let $[n, ChannelOffset]$ denote a link between two nodes. Then the communication frequency (channel) to be used for communication in time slot n of the slotframe is derived as

$$f = F[(ASN + ChannelOffset)\%S_{length}] \tag{1}$$

where ASN is the Absolute Slot Number, defined as the total number of time slots elapsed since the start of the network, and "%" is the modulo operator. F is a lookup table that contains a sequence of available physical channels. S_{length} is the length of the sequence. Please note that S_{length} may be larger than $N_{channel}$, implying that some channels appear multiple times in the table F. The first device in the network sets ASN to 1 and the newcomers learn ASN from the existing devices. Each device in the network increments ASN at every time slot and uses it as a global time slot counter. Eq. 1 defines the TSCH channel hopping mechanism by returning a different channel for the same link ($ChannelOffset$) at different time slots. Many links can transmit simultaneously in the same time slot, provided that they are assigned with different $ChannelOffset$. Figure 1 shows an example where the network consists of four links and uses three channels. Each link has been assigned with a $ChannelOffset$ (0, 1, or 2) that represents channel 24, 25, or 26 in F and each node keeps tracking ASN. In each time slot, the sender and receiver of an active link use Eq. 1 to compute their communication channel. The table in Figure 1 lists the channel hopping sequence for each link. For example, node b and node a use the channel 25 ($(ChannelOffset + ASN)\%3 = 1$) to transmit and receive data in time slot 1 if link $b \rightarrow a$ is active in that slot. The transmission scheduler that runs on top of the MAC layer is responsible for deciding which set of links should be active in each time slot. The underlined numbers in Figure 1 describe an example schedule of active links, which allows node a to collect readings from the rest of the nodes in every four time slots. The IEEE 802.15.4 standard specifies neither any scheduling algorithm nor the way of generating physical channel sequence, but it defines the abovementioned mechanism to execute a schedule provided by the scheduler in the upper layer.

The function-based channel hopping used in TSCH simplifies the network operations because there is no need for the network device to synchronize the channel. In a conventional TSCH network,

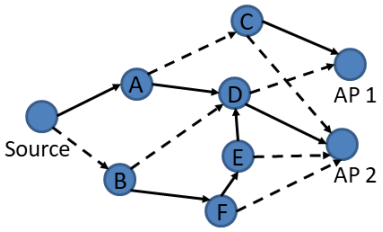


Fig. 2. A graph routing example. The solid lines represent the primary paths and the dashed lines represent the backup paths.

each node learns the current *ASN* and the channels used in the network from its neighbors upon joining the network, and then uses those information to generate a channel hopping sequence, typically until it leaves the network. However, the channel sequences generated by TSCH present a strong pattern, which introduces a security vulnerability to the network. We will present how an attacker derives the channel hopping sequences without knowing any parameter of Eq. 1 in the following section.

WirelessHART adopts graph routing to enhance end-to-end reliability by taking advantage of the route diversity. Graph routing involves a routing graph consisting of a directed list of paths between network devices. Graph routing consists of a single primary path and a backup path for each device. As illustrated in Figure 2, the packet may take backup routes (through device B, C, E, or F) to reach the access points (AP 1 and AP 2) if the links on the primary path (through nodes A and D) fail to deliver a packet. The graph routing specified by WirelessHART requires each node to have at least two outgoing paths.

3 VULNERABILITY ANALYSIS ON TSCH CHANNEL HOPPING

We let $ASN = N_s * N + X$ and rewrite Eq. 1 as

$$f = F[(N_s * N + X + ChannelOffset)\%S_{length}] \tag{2}$$

where N_s is the number of time slots in the combined slotframe[2], N is the number of slotframes elapsed since the start of the network, and X is the time slot offset in the combined slotframe. In this section, we demonstrate how an attacker, without any prior knowledge on the operating network (any parameter of Eq. 2), cracks the channel hopping sequences by silently observing the channel activities.

The attacker is assumed to be a device that is capable of monitoring all transmission activities on all 16 channels in the 2.4 GHz ISM band within its overhearing range and has moderate computational capability (e.g., a Raspberry Pi 3 Model B [31] integrating with a Wi-Spy USB Spectrum Analyzer [52]). Today, many TSCH networks are deployed in open fields to support wireless monitor and control applications (e.g., in oil drilling plants). The attacker may be placed or airdropped into the field and powered by batteries or energy harvesting. The intention of the attacker is to launch selective jamming attacks, where the attacker jams only the transmission of interest on its specific communication channel in its specific time slot. Before launching selective jamming attacks, the attacker first needs to crack the channel hopping sequences by silently observing the transmission activities of nearby devices which are inside its overhearing range. Please note that the IEEE 802.15.4 standard leaves the upper layer protocol to decide whether to encrypt the parameters in Eq. 1 during transmissions. However, leaving it unprotected makes

[2]TSCH allows the upper layer protocol to define more than one type of slotframes. All slotframes are merged into a single combined slotframe for execution at runtime.

the problem trivial and the networks vulnerable to attacks. Therefore, in this paper, we assume that the parameters in Eq. 2 are encrypted and the attacker can only get information from the unencrypted MAC message header[3] and cannot understand the encrypted payload that stores ASN, $ChannelOffset$, and S_{length}.

TSCH allows a transmitter to skip some scheduled transmissions at runtime (e.g., skipping the retransmission if the first attempt succeeds, skipping the routing traffic if no update is needed), significantly increasing the difficulty of cracking. We start our analysis from a basic case where transmitters transmit packets in all scheduled slots (*ideal case*) and then extend our analysis to a realistic case where transmitters may skip some time slots with scheduled transmissions (*realistic traffic*).

3.1 Cracking the Channel Hopping Sequences in the Ideal Case

In this case, we assume that all devices transmit packets in their scheduled time slots and the attacker begins to eavesdrop on the channels in the time slot S_1. The encrypted ASN of slot S_1 is unknown to the attacker. Here are the four steps on how an attacker cracks the channel hopping sequences, identifies the slots with scheduled transmissions in the slotframe, and predicts the future channel usage:

(1) **Grouping the eavesdropped packets**: The attacker snoops the channels and groups the eavesdropped packets based on their source and destination addresses stored in the unencrypted MAC message headers. The attacker then identifies the channel usage sequence of each network device.

(2) **Identifying the least common multiple of N_s and S_{length} (denoted as $LCM(N_s, S_{length})$)**: According to Eq. 2, each network device must use the same channel in the time slot S_1 and $S_{1+LCM(N_s,S_{length})}$, S_2 and $S_{2+LCM(N_s,S_{length})}$, $\cdots$. In other words, the channel hopping sequence used by each device repeats in every $LCM(N_s, S_{length})$ time slots. Based on each network device's channel usage sequence, the attacker identifies its usage repetition cycle and measures its time duration $T_{repetition}$. The attacker can also derive the length of a time slot T_{slot} by measuring the minimum time duration between the start of two consecutive transmissions. Finally, the attacker gets $LCM(N_s, S_{length}) = T_{repetition}/T_{slot}$. Please note that the measured $T_{repetition}$ of some network devices may be less than $LCM(N_s, S_{length}) * T_{slot}$, thus the attacker should use the largest value among all measured $T_{repetition}$.

(3) **Identifying the time slots with scheduled transmissions:** From the eavesdropped transmission activities, the attacker identifies the time slots that are scheduled for transmissions in the slotframe.

(4) **Creating a channel offset table:** The goal of cracking the channel hopping sequences by an attacker is to predict the future channel usage and then perform selective jamming attacks. Thus, there is no need for an attacker to obtain the actual values of F, S_{length}, N, X, and $ChannelOffset$ in Eq. 2. The attacker can assume the time slot S_1 is the first slot in the slotframe and set $N = 0$, and then create a table that pairs each time slot with scheduled transmission (between slot S_1 and $S_{LCM(N_s,S_{length})}$) to a channel for each link.

After deriving $LCM(N_s, S_{length})$, the time slots with scheduled transmissions in the slotframe, and the channel offset table, the attacker knows the exact channel hopping sequence of each link in future, and thus can perform precise strikes to any transmission of interest. The channel hopping sequences can be cracked without error within the bounded time $2 * LCM(N_s, S_{length}) * T_{slot}$[4].

[3]Due to the overhead concern, IEEE 802.15.4e does not require any encryption to the MAC message header.

[4]We assume that the attacker needs to snoop $2 * LCM(N_s, S_{length})$ slots to confirm the channel usage repetition cycle.

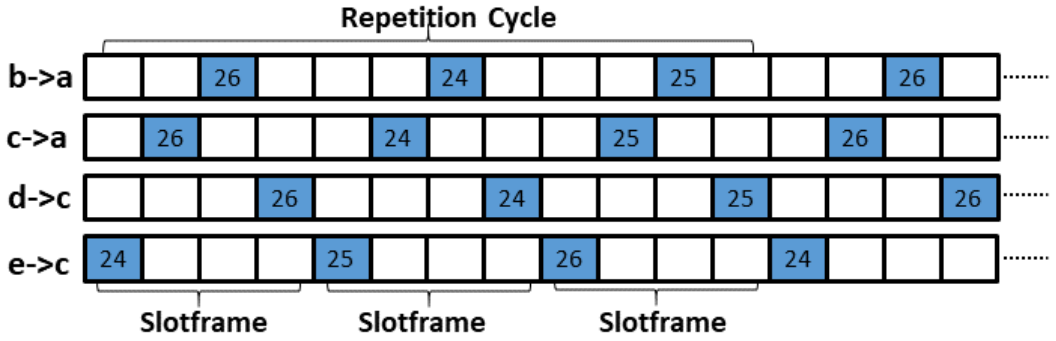


Fig. 3. Section 3.1 example: four slots in the slotframe, three channels (24, 25, and 26) used in the network, and four transmissions scheduled in each slotframe.

Table 2. Channel Offset Table (P increases by 1 every 12 slots).

Link	Slot No.	Channel	Link	Slot No.	Channel
b->a	12*P+3	26	c->a	12*P+2	26
b->a	12*P+7	24	c->a	12*P+6	24
b->a	12*P+11	25	c->a	12*P+10	25
d->c	12*P+4	26	e->c	12*P+1	24
d->c	12*P+8	24	e->c	12*P+5	25
d->c	12*P+12	25	e->c	12*P+9	26

We use an example to illustrate the cracking process. We assume that an attacker is placed in the network presented in Figure 1 and begins to snoop the channels when $ASN = 3$. The attacker does not know ASN and assumes $P = 0$. After snooping for a while, the attacker observes some activities on three channels (channel 24, 25, and 26) and finds that the channel usage repeats in every 12 time slots, as Figure 3 shows. The attacker then derives $LCM(N_s, S_{length}) = 12$ and finds out that all four slots are scheduled with transmissions. Finally, the attacker generates Table 2 and uses it to predict the channel usage for each link in future slots.

3.2 Cracking the Channel Hopping Sequences under Realistic Traffic

Algorithm 1: N_s Identification Algorithm

```
Input  : TSUR[]
Output : N_s
Initialize Density[] to 0 and Position[][] to 0;
for i = 1; i ≤ N_u; i++ do
    for j = 1; j ≤ N_r; j++ do
        if Position[i][TSUR[j]%i] == 0 then
            Position[i][TSUR[j]%i] = 1;
            Density[i] = Density[i] + 1/i;
        end
    end
end
Output k (Density[k] is the smallest value in Density[]);
```

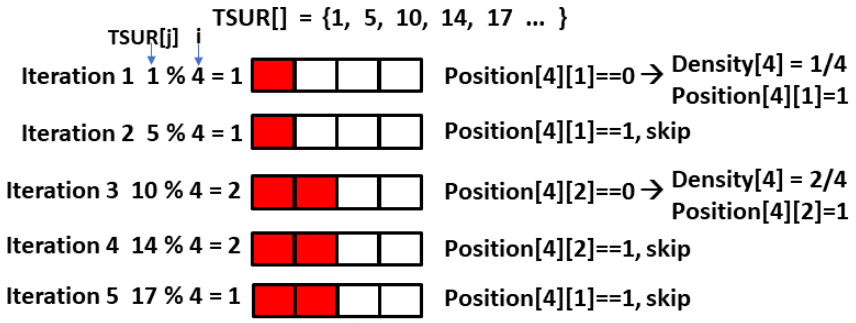


Fig. 4. Algorithm execution example ($i = 4$ and $TSUR[] = \{1, 5, 10, 14, 17, \ldots\}$).

In this case, the attacker cannot easily derive $LCM(N_s, S_{length})$ by identifying the channel usage repetition, because a transmitter may skip some scheduled time slots at runtime, which breaks the repetition pattern. The realistic traffic poses a significant challenge for the attacker to derive N_s, because it cannot easily pinpoint the beginning and the end of a channel usage repetition cycle. However, we find that the attacker is able to accomplish the cracking by employing a "trial-and-error" learning method. Algorithm 1 shows the method that derives N_s. Please note that the attacker does not know ASN because it does not know when the network starts. Instead of using ASN, the attacker defines OSN (Observation Slot Number) and sets the first observed time slot S_1 with $OSN = 1$. If the attacker observes the actual transmission in a time slot, it adds its OSN value into the array, named $TSUR$ (Time Slot Usage Record). The $TSUR$ array is the input of Algorithm 1. N_r denotes the number of usage records that the attacker collects (number of elements in $TSUR[]$). Algorithm 1 first defines a one-dimensional array $Density$ and a two-dimensional array $Position$ with initial values (line 1). Each element $Density[i]$ stores the weighting factor and helps the attacker to identify the likelihood of i being N_s. Each element $Position[i][j]$ indicates whether the time slot with offset j is scheduled for transmission by assuming $i = N_s$.

The two-level nested loop computes the $Density$ value for each possible value of N_s according to $TSUR[]$ (line 2 – 9). The outside loop traverses all possible values of N_s (from 1 to N_u), where N_u is the upper bound of N_s. The inside loop traverses all records in $TSUR[]$ (from 1 to N_r). At each iteration, Algorithm 1 marks the time slots with scheduled transmissions with 1 in the $Position[i][]$ array by assuming $i = N_s$ (line 4 – 7). $TSUR[j]\%i$ indicates the corresponding slot offset for the record $TSUR[j]$ when applying it to a slotframe that consists of i slots. If the offset ($Position[i][TSUR[j]\%i]$) has not been previously labeled by any record, Algorithm 1 marks it with 1 (line 5) and increases $Density[i]$ by 1/i (line 6). As an example, Figure 4 illustrates the first five iterations of the inside loop when $i = 4$ in Algorithm 1. In the first iteration, $TSUR[1] = 1$, so the condition $Position[4][1\%4] == 0$ is met. $Position[4][1]$ is then set to 1 and $Density[4]$ increases by $1/i = 1/4$. In the second iteration, $TSUR[2] = 5$, so the condition $Position[4][5\%4] == 0$ is not met. $Density[4]$ does not change. Similarly, $Density[4]$ increases by $1/4$ in the third iteration and does not change in the fourth and fifth iterations. After the outside loop exits, Algorithm 1 outputs the array index with the smallest value in $Density$ array. The index is either N_s or a multiple of N_s[5]. In other words, the output of Algorithm 1 $\in \{m * N_s | m \in \mathbb{N}^+\}$. We prove the statement by contradiction.

Proof. We assume that there exists n ($n\%N_s \neq 0$) and $Density[n] < Density[m * N_s]$ ($\forall m \in \mathbb{N}^+$) and separate the proof into two cases: (1) N_s and n do not share any common factor and (2) N_s and n share at least one common factor.

[5]Using N_s or a multiple of N_s to generate the channel offset table are functionally equivalent for the attacker. The only difference is the size of channel offset table.

Case 1: We assume that the first element in the $Position[][]$ array marked as 1 by Algorithm 1 when $i = n$ is $Position[n][p]$. Since N_s and n do not share any common factor, Algorithm 1 marks $Position[n][p]$, $Position[n][(p + N_s)\%n]$, ..., $Position[n][(p + (n-1)N_s)\%n]$ as 1 after executing the line 3–8 if there are sufficient observations in $TSUR[]$. After Algorithm 1 exits, the first n elements of $Position[n][]$ are all marked as 1. So we have $Density[n] = 100\% \geq Density[N_s]$, contradicting the assumption.

Case 2: We assume that there exists n that shares at least one common factor with N_s and the largest common factor (LCF) of N_s and n is $LCF(N_s, n)$. We divide the slotframe consisting of N_s slots into $\frac{N_s}{LCF(N_s,n)}$ blocks, each of which has $LCF(N_s, n)$ slots. We then divide the slotframe into $\frac{n}{LCF(N_s,n)}$ blocks, each of which has $LCF(N_s, n)$ slots. We define the densities of the blocks in the slotframe with N_s slots as: $\rho_1, \rho_2, \rho_3, ..., \rho_{\frac{N_s}{LCF(N_s,n)}}$. Similarly, we define the densities of the blocks in the slotframe with n slots as: $\rho'_1, \rho'_2, \rho'_3, ..., \rho'_{\frac{n}{LCF(N_s,n)}}$. We now treat each block as a single unit. $\frac{N_s}{LCF(N_s,n)}$ and $\frac{n}{LCF(N_s,n)}$ do not share any common factor. So we can convert Case 2 into Case 1. In the fourth line of Algorithm 1, when $i = n$, it maps each element in $TSUR[]$ to a block in the slotframe consisting of n slots ($TSUR[]\%n$). According to the proof for Case 1, all $\frac{n}{LCF(N_s,n)}$ blocks are eventually marked by all blocks in the slotframe consisting of N_s slots after Algorithm 1 finishes executing the line 3–8 if there are sufficient observations in $TSUR[]$. So we have

$$\rho'_x \geq max\{\rho_1, \rho_2, \rho_3, ..., \rho_{\frac{N_s}{LCF(N_s,n)}}\} \ \ (\forall x \in [1, 2, 3, ..., \frac{n}{LCF(N_s,n)}])$$

and then the density of the slotframe when assuming the slotframe has n slots is

$$\begin{aligned} Density[n] &= \frac{\rho'_1 + \rho'_2 + \rho'_3 + ... + \rho'_{\frac{n}{LCF(N_s,n)}}}{\frac{n}{LCF(N_s,n)}} \\ &\geq \frac{n}{LCF(N_s,n)} \times \frac{max\{\rho_1, \rho_2, \rho_3, ..., \rho_{\frac{N_s}{LCF(N_s,n)}}\}}{\frac{n}{LCF(N_s,n)}} \\ &= max\{\rho_1, \rho_2, \rho_3, ..., \rho_{\frac{N_s}{LCF(N_s,n)}}\} \\ &\geq \frac{\rho_1 + \rho_2 + \rho_3 + ... + \rho_{\frac{N_s}{LCF(N_s,n)}}}{\frac{N_s}{LCF(N_s,n)}} = Density[N_s] \end{aligned}$$

So we have $Density[n] \geq Density[N_s]$, contradicting the assumption.

With the proof for both cases, we finish the proof by contradiction. □

The time complexity of Algorithm 1 to derive N_s is $O(N_u * N_r)$. If the attacker uses a constant N_u, the time complexity becomes $O(N_r)$. Please note that there is no need to rerun Algorithm 1 when the attacking program obtains new records. The attacking program can only take the new records as input and process them (executing line 2 – 9) based on the existing *Density* array and *Position* array.

After obtaining N_s (or its multiple), the attacker can identify the repetition cycle $LCM(N_s, S_{length})$ by exploring all possible S_{length} values. If every two transmissions with a time interval of $LCM(N_s, M)$ slots always use the same channel, $LCM(N_s, M)$ can be used as the repetition cycle. The attacker then follows the same methods presented in Section 3.1 to identify the time slots with scheduled transmissions, and generate the channel offset table. Please note that S_{length} may change at runtime when a channel is excluded from the network or added into the network due to channel condition changes. The attacker needs to keep monitoring the channel usage when launching jamming attacks. When detecting a channel usage change, the attacker needs to repeat the above process to identify the new $LCM(N_s, S_{length})$.

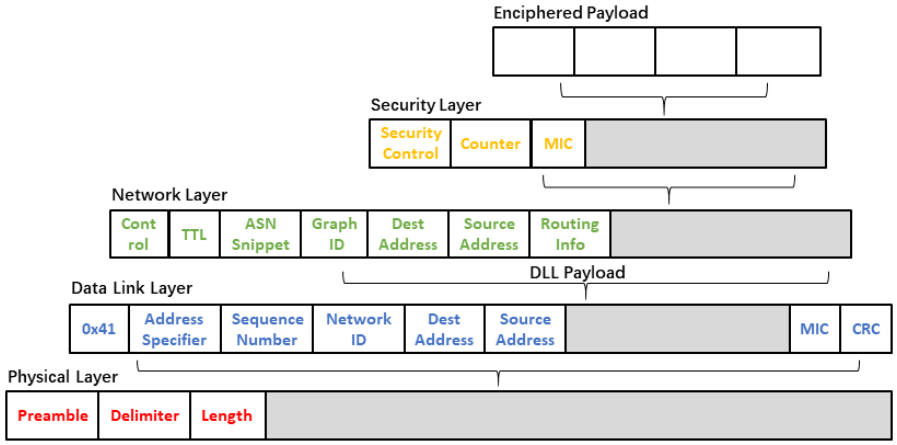


Fig. 5. Data DLPDU structure of WirelessHART.

4 VULNERABILITY ANALYSIS ON GRAPH ROUTING

In this section, we present our vulnerability analysis on graph routing in WirelessHART networks by demonstrating how an attacker cracks the graph routes by silently observing the packet transmission activities.

In WirelessHART networks, a data-link protocol data unit (DLPDU) provides means for reliable communication in the data-link layer (DLL). As Figure 5 shows, a WirelessHART DLPDU consists of sequence number, network ID, destination address, source address, DLL payload, message integrity code (MIC), and some other fields. A network protocol data unit (NPDU) is stored in the DLL payload, which is composed of protocol specific information and user data. WirelessHART does not require the network to encrypt the DLPDU and NPDU headers due to the overhead concern. For example, a recent study [61] shows that encrypting the packet headers following Advanced Encryption Standard (AES) takes 1.1ms on a CC2420 device, which is too long to be fit into a 10ms time slot used in WirelessHART. The source and destination addresses of a communicating link are stored in the DLPDU header, while the address of the device which originally generated the packet and the final destination address of the packet are stored in the NPDU header. The attacker may make use of the unencrypted information stored in the eavesdropped packets to derive the graph routes. Specifically, the attacker can execute the following four steps to crack the routes used by the graph routing in WirelessHART networks:

(1) **Grouping the eavesdropped packets**: The attacker snoops the packet transmission activities and groups the eavesdropped packets based on their original source and final destination addresses carried in the unencrypted NPDU headers. All packets that share the same original source and final destination addresses in the NPDU headers belong to the same data flow.
(2) **Sorting the packets by their capture time:** In each group of packets, the attacker sorts the packets by their time stamps at capture.
(3) **Identifying the primary routing path for each data flow:** In each group of sorted packets, the attacker identifies the primary routing path of each data flow by comparing the addresses storing in the DLPDU header with the ones in the NPDU header. As each DLPDU header includes the source and destination addresses for the communicating link, the attacker can identify all intermediate devices located on the primary routing path by checking the sorted packets one by one until the link destination address in the DLPDU header is the same as the path destination address in the NPDU header carried by the DLPDU.

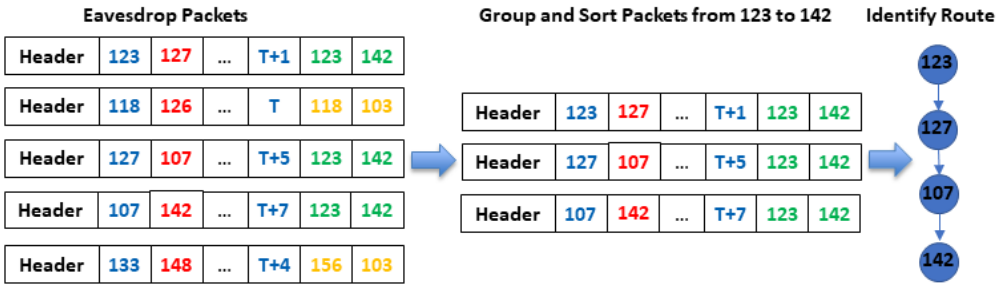


Fig. 6. Example of cracking process.

Table 3. Example of Cracking Result

Source	Destination	Routing Path	Hops
123	142	$123 \xrightarrow{T+1} 127 \xrightarrow{T+5} 107 \xrightarrow{T+7} 142$	3

(4) **Identifying the backup routes:** As each device located on the primary routing path may transmit packets through its backup route, the attacker can identify those backup routes by selectively jamming each link on the primary path.

Figure 6 uses an example to illustrate the cracking process. We assume that an attacker is placed in the network and eavesdrops the transmission activities. Since the attacker does not know *ASN*, it uses $T = 0$ as the starting point and assigns a time offset to each eavesdropped packet according to its capture time. Then the attacker selects the packets with the original source address 123 and the final destination address 142, and sorts those packets by their capture time. Beginning from the first eavesdropped packet with the same original source address and link source address (123), the attacker identifies all intermediate nodes until it reaches the packet with the same final destination address and link destination address (142). Finally, the attacker creates a table for the primary routing path from node 123 to node 142, which includes each hop and its corresponding time slot, as Table 3 lists. The time slots assigned to the transmissions $123 \rightarrow 127$, $127 \rightarrow 107$, and $107 \rightarrow 142$ are $T + 1$, $T + 5$, and $T + 7$, respectively. After deriving the primary routing path, the attacker cracks the backup routes by jamming the transmission on each link located on the primary routing path, repeating step (1), (2), and (3) to obtain the new routing paths, and then compare them against the original ones. Then the attacker generates the table listing the backup route of each node.

With the cracked TSCH channel hopping sequences and graph routes, the attacker is able to launch the selective jamming attacks by jamming only the transmission of interest on its specific communication channel in its specific time slot. Please note that the attacker can predict the channel usage of any device since all devices in the network uses the same parameters in Eq. 2 to generate the channel hopping sequences, but it may not be able to observe the transmission activities of devices far away, thus it may not derive all routes used in the network.

5 CASE STUDY ON ORCHESTRA

In this section, we present our case study on cracking the channel hopping sequence of the TSCH implementation [12] in Contiki operating system [11] developed for Orchestra [14] and 6TiSCH networks [18][6]. Orchestra proposes an autonomous transmission scheduling method running on top of the IPv6 Routing Protocol for Low-power and Lossy Networks (RPL) [42] and TSCH networks. Each node computes its transmission schedule locally based on its routing state and MAC address. All nodes running Orchestra change the channels together following the TSCH channel hopping

[6]The implementation is provided by Duquennoy et al. and is publicly accessible [13].

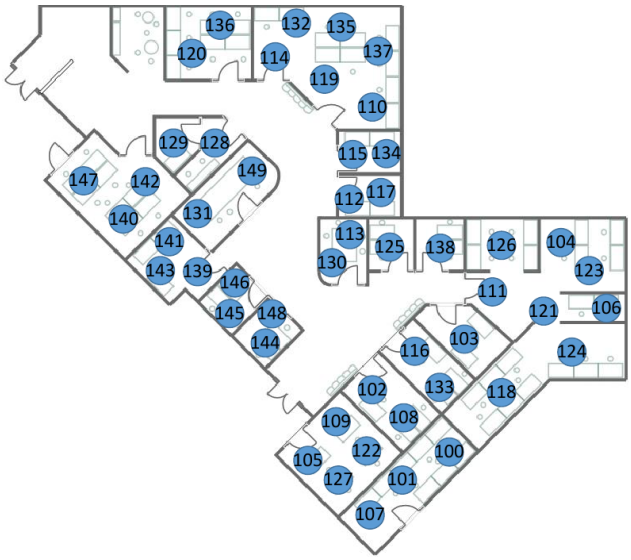


Fig. 7. Testbed consisting of 50 TelosB motes.

method (Eq. 2). Orchestra employs three types of slotframes for three different kinds of traffic: application, routing, and time synchronization. Different types of slotframes are assigned with different lengths. Orchestra allows S_{length} to be larger than $N_{channel}$.

5.1 Experimental Methodology

We run the experiments on a testbed that consists of 50 TelosB motes [47] deployed on a single floor of an office building [44]. Figure 7 plots the testbed topology. We configure the network to have a single access point and 49 network devices operating on four channels (the default value in Orchestra). The slotframe lengths for application, routing, and time synchronization are 47, 31, and 61, respectively. The combined slotframe has 88,877 time slots in total. Each network device generates a packet every 20s. The attacking program runs on a Raspberry Pi equipped with a 1.2GHz 64-bit quad-core processor and 1.0 GB memory. We perform three sets of experiments. We first measure the prediction performance and cracking time when the attacker snoops different amount of time before launching the attack. We then examine the impact of slotframe length on the cracking performance. Finally, we study the cracking performance when employing a tailored version of the attacking program to Orchestra. We record all the channel activities during the experiments and use them as the the ground truth.

5.2 Cracking Performance with Different Snooping Periods

We configure the attacking program to start cracking after snooping the channel activities during a certain number of time slots (snooping period). We vary the length of snooping period from 88,877 slots (1 combined slotframe) to 2,133,048 slots (24 combined slotframes). The channel usage during the snooping period is used as the training set and the channel usage of the next 1,599,786 slots (18 combined slotframes) is taken as the validation set. Our cracking program provides predicted N_s and $LCM(N_s, S_{length})$, identifies the future slots with scheduled transmissions, and predicts the channels used by future transmissions. We compare the predicted transmission activities and their channels against the ground truth in the validation set. If the predicted time slots with transmissions (and corresponding channels) and the ones without transmissions match the ground truth, they

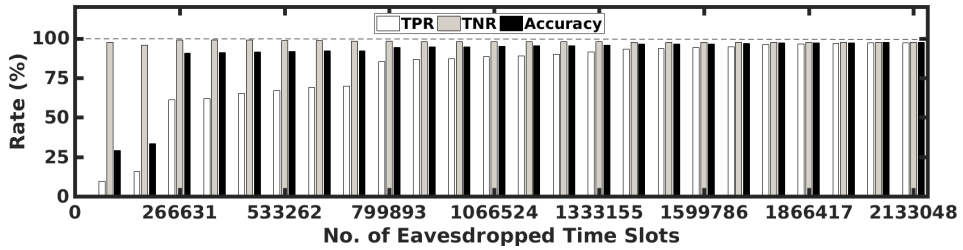


Fig. 8. Cracking performance on Orchestra with different snooping periods.

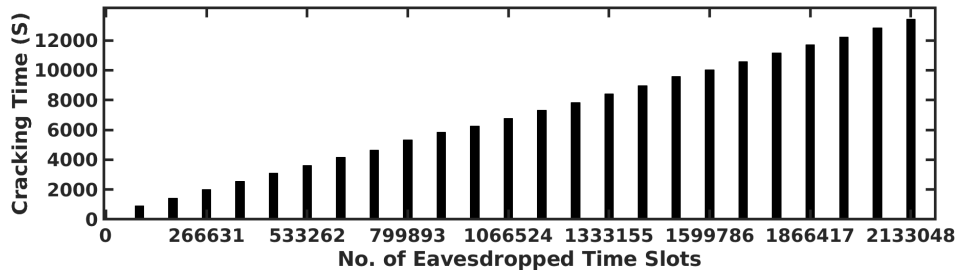


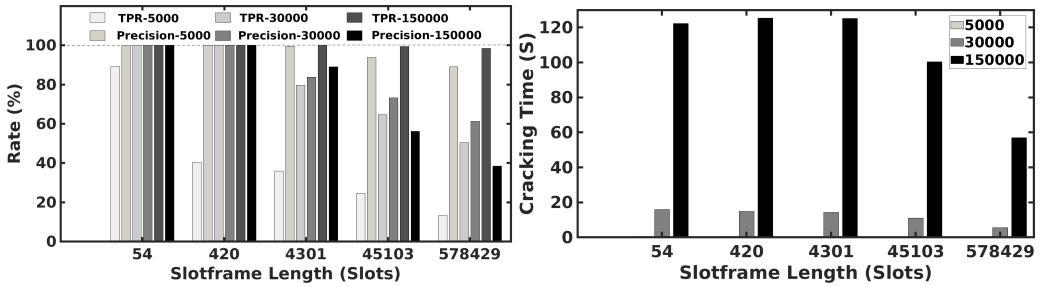
Fig. 9. Cracking time with different length of snooping.

are labeled as True Positive (TP) and True Negative (TN), respectively. The wrong predictions are marked as False Positive (FP) and False Negative (FN). After labeling all predictions, we compute the True Positive Rate ($TPR = TP/(TP + FN)$), True Negative Rate ($TNR = TN/(TN + FP)$), Accuracy ($Accuracy = (TP + TN)/(TP + FN + FP + TN)$), and Precision ($Precision = TP/(TP + FP)$).

Figure 8 plots TPR, TNR, and Accuracy of the predictions with different amount of training data (snooping period). As Figure 8 shows, TPR and Accuracy are small (9.65% and 15.55% for TPR, 29.03% and 33.22% for Accuracy) when the eavesdropped number of slots are 88,877 and 177,754 (first two sets of bars). Without enough observations, the cracking program provides a wrong N_s, making the predictions very inaccurate. TPR and Accuracy increase sharply to 60.91% and 90.65%, respectively, when there are 266,631 slots (3 combined slotframes) in the training set. Although the predicted N_s provided by the cracking program is still incorrect, it shares a common factor with the actual value, resulting in some correct prediction on the future channel usage. TPR and Accuracy then increase slowly when the training set is increasing from 266,631 to 711,016 eavesdropped time slots. TPR and Accuracy reach 85.15% and 94.18% with 799,893 eavesdropped slots, providing accurate prediction on the channel usage. This is because the training set includes enough observations for the attacking program to produce the correct N_s leading to accurate channel usage prediction. After that, the increases of TPR and Accuracy become moderate when the training set is larger than 1,777,540 slots (TPR ranging from 96.09% to 97.20% and Accuracy ranging from 97.03% to 97.31%). We observe a similar trend on TNR.

Figure 9 shows the time consumed by the attacking program to crack the channel hopping sequence[7]. The time consumption increases linearly from 924s (88,877 slots) to 13430s (2,133,048 slots), which accords with the $O(N_r)$ time complexity of Algorithm 1. Please note that the attacking program spends most of time on cracking N_s. N_s is not expected to change at runtime, since it is selected based on the data rates, which are often fixed for a given control application. Compared to the time spent on deriving N_s, the time spent on identifying S_{length} is very short. When S_{length} changes, it only takes a short time period to identify the new value by observing the channels used in the network.

[7]The snooping period is not added into the result.



(a) TPR and Precision. (b) Time Consumption.

Fig. 10. Cracking performance with different slotframe length.

Table 4. Slotframe Composition.

No.	Syn	Routing	App	Product	LCM
1	27	9	18	4374	54
2	21	10	20	4200	420
3	23	11	17	4301	4301
4	53	23	37	45103	45103
5	397	31	47	578429	578429

Observation 1*: An attacker can predict the TSCH channel hopping sequences accurately under realistic traffic.*

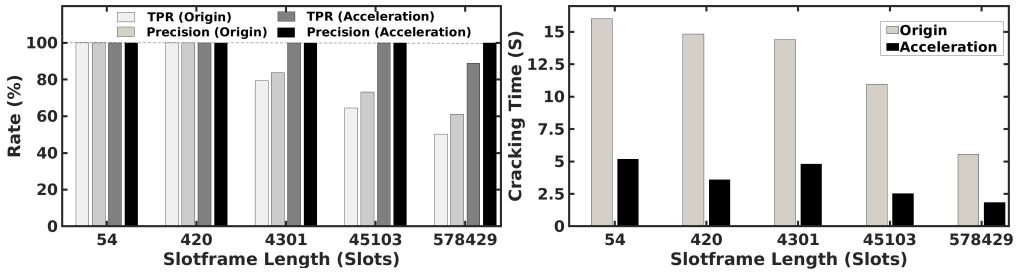
Observation 2*: A sudden increase on prediction accuracy does not warrant a correct predicted N_s.*

Observation 3*: The attacker can predict the channel usage accurately when the observations are large enough to derive N_s. After that, more observations are desired to further improve the prediction accuracy with small additional value.*

5.3 Impact of Slotframe Length

To explore the impact of slotframe length on the cracking difficulty, we perform five sets of experiments and increase the length of the combined slotframe roughly 10 times for each set (from 54 slots to 578,429 slots). Table 4 lists the number of slots in each type of slotframe as well as the product and LCM of them. In each set of experiments, we run the experiments three times with different amount of training data, namely eavesdropped with 5,000, 30,000, and 150,000 time slots. We use the next 45,000 slots for validation.

Figure 10(a) plots TPR and Precision of the predictions with different combined slotframe lengths. As Figure 10(a) shows, TPR and Precision decrease when the slotframe length becomes larger. For example, with 5,000 eavesdropped time slots in the training data, TPR and Precision are high (89.19% for TPR, 100% for Precision) while the slotframe length is 54 slots. The slotframe length is so short that such few observations are enough to identify the channel usage repetition and provide a correct N_s. TPR decreases sharply to 40.16% when the slotframe length is 420 slots. Since the predicted N_S shares a common factor (210) with the correct one, Precision still can reach 100%. With the slotframe length increasing from 4,301 to 578,429 slots, TPR and Precision decrease significantly (TPR dropping from 35.75% to 13.12%, Precision dropping from 99.49% to 89.04%), indicating that the eavesdropped slots are too short to identify a complete channel usage repetition cycle. Similarly, with 30,000 eavesdropped slots in the training data, both TPR and Precision are 100% when the slotframe includes 54 or 420 slots. This is because the eavesdropped activities are enough for



(a) TPR, Precision Comparison. (b) Time Consumption Comparison.

Fig. 11. Cracking performance after removing the strong repetitions.

Algorithm 1 to produce the correct N_s and identify all scheduled slots. After the slotframe length reaches 4,301 slots, TPR and Precision experience a significant decrease (TPR ranging from 79.48% to 50.28%, Precision ranging from 83.69% to 61.13%), indicating the eavesdropped slots are too short to identify a complete repetition cycle. A similar trend is observed when there are 150,000 eavesdropped slots in the training data.

Within each group of bars in Figure 10(a), TPR increases with the training data size. For example, TPR increases from 35.75% (5,000 eavesdropped slots) to 79.48% (30,000 eavesdropped slots), and finally becomes 100% (150,000 eavesdropped slots), when the slotframe length is 4,301 slots. 5,000 eavesdropped slots are too short for Algorithm 1 to pinpoint a repetition cycle, while 30,000 eavesdropped slots are enough to provide the correct N_s. When the training set includes 150,000 eavesdropped slots, it is large enough to identify all slots with scheduled transmissions. Due to insufficient observations, Precision decreases while TPR increases for some group of bars. Precision drops from 89.04% (5,000 slots) to 61.13% (30,000 slots), reaches 38.31% (150,000 slots) when the slotframe length is 578,429 slots. The observations are insufficient for the cracking program, so TPR increases at the cost of generating more FP, making Precision decrease dramatically.

Figure 10(b) presents the time consumed by the attacking program to crack the channel hopping sequences. In each set of experiments, the time consumption increases approximately linearly with the increase of eavesdropped time slots, confirming the time complexity of Algorithm 1. For instance, the time consumption increases from 0.55s (5,000 slots) to 16.01s (30,000 slots), then to 122.20s (150,000 slots) when the slotframe length is 54 slots.

Observation 4*: The combined slotframe length plays an important role in keeping the channel hopping sequence unpredictable. A larger slotframe significantly increases the cracking difficulty.*

Observation 5*: It is beneficial to use a prime number for each slotframe length, which effectively enlarges the combined slotframe.*

Observation 6*: TPR may increase at the cost of decreasing Precision when the snooping period is smaller than the slotframe length. A low precision caused by insufficient observations may expose the attacker during jamming.*

5.4 Impact of Strong Transmission Pattern

When performing the above experiments, we observe that there exist some time slots showing strong cyclic patterns of transmissions, which help the attacking program to identify the repetition cycles. The transmissions cyclic behavior is introduced by the scheduling design in Orchestra. For example, a fixed and shared slot in the routing slotframe is assigned for all network devices to exchange routing related packets including the DODAG Information Object (DIO) and Destination Advertisement Object (DAO) messages and the device i uses the ith slot in the synchronization slotframe to broadcast beacons and jth slot to receive beacons from its parent (device j). After

observing the patterns, the attacking program can first extract the channel activities with these patterns from the observations and then perform the cracking. We repeat the experiments when applying this tailored attacking method. Figure 11(a) compares TPR and Precision between the original attacking program (Origin) and the tailored version (Acceleration) when the length of the training set is 30,000 slots. TPR and Precision are 100% for both methods when the slotframe length is 54 and 420 slots. This is because these observations are enough for both methods to derive the correct N_s. TPR and Precision of Acceleration are much higher than Origin (100% and 100% for Acceleration, 79.48% and 83.69% for Origin) when the slotframe includes 4,301 slots. TPR and Precision of Acceleration are also much higher when the slotframe lengths are 45,103 and 578,429 slots. Figure 11(b) compares the time consumption used by each method with 30,000 eavesdropped slots. For each slotframe length, the time spent for cracking decreases by more than 60% (e.g., from 5.55s to 1.85s for 578,429 slots) after acceleration. Benefiting from the extraction, the actual combined slotframe length decreases dramatically. The attacking program therefore produces better cracking performance with less time consumption.

***Observation 7**: The strong cyclic behavior of packet transmissions significantly reduces the cracking difficulty.*

5.5 Suggestions to Orchestra

Here are some suggestions gathered from our case study:

- It is beneficial to set three different prime numbers as the lengths of the three slotframes in Orchestra. So the length of the combined slotframe is large (LCM of those numbers), significantly increasing the cracking difficulty. The larger a combined slotframe is, the more time an attacker have to spend on snooping and cracking. Orchestra's decision on using prime numbers as the lengths of the slotframes significantly enhances the network security.
- It is beneficial to use multiple slotframes for application traffic. So the length of the combined slotframe is enlarged (LCM of those numbers), which significantly increases the cracking difficulty.
- It is beneficial to randomize the distribution of the slots with transmissions in each slotframe for routing and time synchronization. The strong cyclic behavior of packet transmissions greatly reduces the difficulty of cracking.

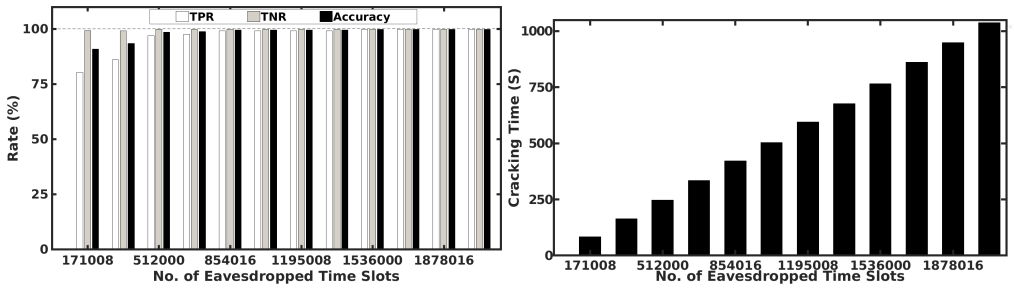
6 CASE STUDY ON WIRELESSHART

In this section, we present our case study on cracking the channel hopping sequence and graph routes of the TSCH implementation in TinyOS operating system [22] developed for WirelessHART networks[8] [23–25]. Typically, a WirelessHART network consists of a gateway, multiple access points, and a set of field devices (sensors and actuators) forming a multi-hop mesh network. The network is managed by a centralized network manager, a software module running on the gateway, which is responsible for generating routes and transmission schedules and maintaining the operation of the network. Different from Orchestra, all devices in the WirelessHART networks follow the channel hopping sequences generated by the network manager. To enhance the network utilization, the network manager assigns different $ChannelOffsets$ to different links, allowing up to $N_{channel}$ packets to be transmitted simultaneously in each time slot. WirelessHART supports both source and graph routing. Source routing provides a single route for each data flow, whereas graph routing generates a reliable routing graph in which each device should have at least two neighbors to which they can forward packets.

[8]The implementation is provided by Li et al. and is publicly accessible [51].

Table 5. Six data flows configured in WirelessHART network.

Flow	Sensor	Actuator	Period	Priority
1	147	146	320ms	1
2	144	143	640ms	2
3	105	104	1280ms	3
4	149	102	2560ms	4
5	136	135	5120ms	5
6	137	108	10240ms	6



(a) TPR, TNR, and Accuracy.

(b) Time consumption.

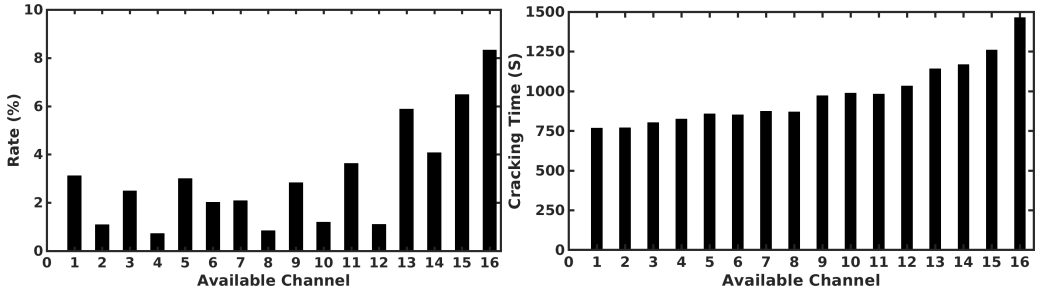
Fig. 12. Cracking performance on WirelessHART with different snooping periods.

6.1 Experiment Methodology

We run the experiments on the same testbed and configure the network to have two access points and 48 field devices operating on four channels. As Table 5 lists, we set up six data flows with different sources, destinations, data periods, and priorities and employ graph routing as well as the rate monotonic policy for transmission scheduling. The slotframe lengths are 32, 64, 128, 256, 512 and 1,024 as suggested in the WirelessHART standard [54]. Therefore, the combined slotframe includes 1,024 time slots in total. A maximum of three transmission attempts are scheduled for each packet. The first two attempts go through the primary route and the final attempt uses the backup route in the routing graph. WirelessHART [54] specifies that S_{length} is equal to $N_{channel}$ and every channel appears one time in F. We perform four sets of experiments. First, we measure the prediction performance and cracking time when the attacker snoops different amount of time before launching the attack. Second, we vary S_{length} and investigate its impact on the cracking difficulty. Then we measure the snooping and cracking time used by an attacker to derive the graph routes used in the network when operating in different environments. Finally, we vary the number of hops of a data flow and investigate its impact on the cracking performance.

6.2 Cracking Performance with Different Snooping Periods

In this set of experiments, we vary the size of the training set (number of eavesdropped time slots) from 171,008 slots (167 combined slotframes) to 2,048,000 slots (2,000 combined slotframes) and use the channel activities during the next 1,707,008 slots (1667 combined slotframes) as the validation set. Figure 12(a) plots TPR, TNR, and Accuracy. As Figure 12(a) presents, TPR and Accuracy are low (80.30% and 86.09% for TPR, 90.95% and 93.38% for Accuracy) when 171,008 and 342,016 time slots are eavesdropped (167 and 334 combined slotframes as presented as the first two sets of bars). Without enough observations, the cracking program fails to derive the correct N_s. However, the N_s



(a) FPR. (b) Time consumption.

Fig. 13. Cracking performance with different S_{length} in the network.

produced by the attacking program shares a common factor with the actual value, resulting in some correct predictions on the future channel usage (TPRs higher than 80%). Comparing Figure 12(a) and Figure 8, we observe that the attacking program achieves much higher TPR and Accuracy on WirelessHART than those on Orchestra with a similar amount of eavesdropped time slots which is insufficient for Algorithm 1 to derive the correct N_s. This is because the number of slots in each slotframe is specified to be 2^n in WirelessHART, resulting in a smaller combined slotframe and a short repetition cycle. In contrast, the default slotframe lengths of Orchestra are prime numbers, leading to a significant larger slotframe. Therefore, it is harder for the attacker to capture the repetition cycle. As Figure 12(a) shows, TPR and Accuracy experience a quick rise when 512,000 slots has been eavesdropped. TPR and Accuracy reach 96.98% and 98.54%, providing very accurate predictions on the channel usage. The training set is large enough for the attacking program to derive the correct N_s and predict the channel usage. TPR and Accuracy then increase slowly when the training set becomes larger (TPR ranging from 96.98% to 99.78% and Accuracy ranging from 98.54% to 99.80%). We observe consistent high TNRs, since most time slots in the slotframe are not scheduled with transmissions.

Figure 12(b) shows the time consumed by the attacking program to crack the channel hopping sequence[9]. The time consumption increases linearly from 83.18s (171,008 slots) to 1037.47s (2,048,000 slots), which accords with the $O(N_r)$ time complexity of Algorithm 1.

Observation 8*: The cracking difficulty depends highly on the length of combined slotframe (LCM of different slotframe lengths). The attacking program consumes more time when cracking larger combined slotframe but provides less accurate predictions.*

6.3 Impact of the Length of Sequence S_{length}

In this set of experiments, we increase S_{length} from 1 to 16 and repeat the experiments. For all experimental executions, we configure the attacking program to crack after snooping the channel activities for 1,366,016 time slots (1,334 combined slotframes) and use the following 1,366,016 slots for validation. Figure 13(a) plots False Positive Rate ($FPR = FP/(FP + TN)$). As Figure 13(a) shows, FPRs are 5.88%, 4.08%, 6.49%, and 8.33% when 13∼16 channels are available in the network, higher than the ones with less channels. Figure 13(b) plots the time consumed by the attacking program to crack the channel hopping sequences with different S_{length}. The time consumption increases from 768.53s when the network uses one channel, to 973.15s when the network uses nine channels, and finally reaches 1465.03s when the network uses all 16 channels. The results show that the cracking becomes more difficult when more channels are used in the network. This is because the data flows

[9]The snooping period is not added into the result.

Table 6. Primary Path

Flow	Routing Path and Time Slot	Hops
1	$147 \xrightarrow{T} 139 \xrightarrow{T+2} 113 \xrightarrow{T+4} 103 \xrightarrow{T+6} 121 \xrightarrow{T+12} 126 \xrightarrow{T+14} 146$	6
2	$144 \xrightarrow{T} 108 \xrightarrow{T+2} 121 \xrightarrow{T+15} 108 \xrightarrow{T+17} 144 \xrightarrow{T+19} 143$	5
3	$105 \xrightarrow{T} 109 \xrightarrow{T+3} 101 \xrightarrow{T+18} 121 \xrightarrow{T+24} 104$	4
4	$149 \xrightarrow{T} 113 \xrightarrow{T+11} 103 \xrightarrow{T+22} 121 \xrightarrow{T+31} 101 \xrightarrow{T+33} 102$	5
5	$136 \xrightarrow{T+7} 114 \xrightarrow{T+9} 110 \xrightarrow{T+14} 113 \xrightarrow{T+18} 103 \xrightarrow{T+34} 121 \xrightarrow{T+42} 103 \xrightarrow{T+44} 113 \xrightarrow{T+46} 110 \xrightarrow{T+48} 135$	9
6	$137 \xrightarrow{T+12} 110 \xrightarrow{T+25} 113 \xrightarrow{T+30} 103 \xrightarrow{T+47} 121 \xrightarrow{T+53} 108$	5

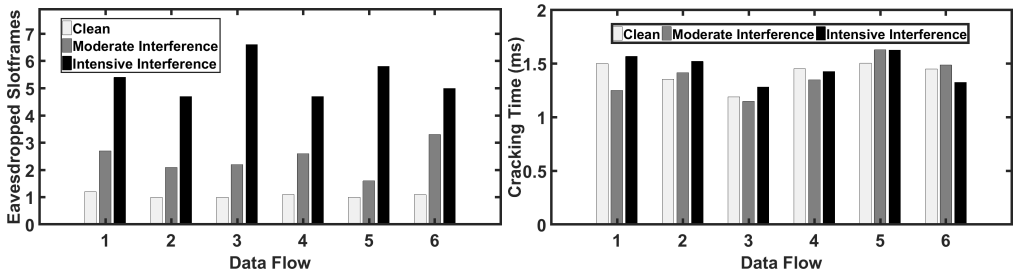
involve more hops when more channels are available for use [16], resulting in more transmissions in each slotframe.

Observation 9*: It is difficult to crack the channel hopping sequences when a large S_{length} is used in WirelessHART networks.*

6.4 Cracking Graph Routes in Different Operating Environments

In this set of experiments, we set the attacking program to start cracking after snooping the transmission activities during a certain number of time slots (snooping time) and measure the number of eavesdropped time slots and cracking time consumed by the attacking program to crack the graph routes. We create three different wireless operating environments (clean, moderate interference, and intensive interference) by using JamLab [5] to generate controlled interference with different jamming strengths and repeat the experiments in each environment. Our attacking program derives the routes, identifies the time slots assigned for those routes, and provides the snooping and cracking time. Then we run the attacking program to derive the backup routes by jamming the transmissions through the primary routing paths. We repeat the experiments 20 times for each data flow in each environment and measure the accuracy and time consumption. The cracking program achieves 100% accuracy in all environments. Table 6 shows the primary routing paths and the total number of hops on each data flow cracked by our attacking program. Assuming T as the first eavesdropped time slot, the attacking program marks the time slots to each link between two nodes, as the arrows in Table 6 show. For instance, the attacking program finds that the primary routing path from node 137 to node 108 consists of 6 nodes and 5 hops, which are scheduled in time slot $T + 12$, $T + 25$, $T + 30$, $T + 47$, and $T + 53$, respectively. Figure 14(a) plots the time consumed by the attacking program to snoop the channels and gather enough packets for successfully deriving the primary routing paths. As Figure 14(a) shows, it takes more time to eavesdrop enough packets when the interference is stronger. For example, the attacking program needs an average time period of 1.2 slotframes to derive the primary path for data flow 1 in the clean environment. In a clean environment, source nodes can easily deliver packets to their destinations through the primary paths in the first attempt. With the presence of moderate interference, source nodes may have to perform the second or third attempt using the backup routes. The attacking program needs more observations (2.7 slotframes) to exclude such activities to identify the transmission activities of nodes located on the primary routing path. Under intensive interference, the attacking program consumes an average time period of 5.4 slotframes to crack the graph routes, because most source nodes cannot deliver packets in the first attempt. We observe similar trends in other groups of bars.

Figure 14(b) presents the execution time of our attacking program. The time consumption fluctuates around 1.5ms under each condition for each data flow. The cracking time for data flow 5



(a) Snooping Time. (b) Cracking Time.

Fig. 14. Cracking performance under different interference .

is longer than that of other data flows, because the cracking time mainly depends on the time used to identify the intermediate nodes. There are more hops on data flow 5.

After identifying the primary routing path, we derive the backup routes for each node located on the primary routing path by jamming the transmissions on each link. We take the cracking result of data flow 6 as an example to illustrate the process. The routing path shown in Figure 15(a) is the primary path of data flow 6. When node 137 can not deliver packets to node 110 because of the effect of jamming, it sends packets to node 114 which forwards packets to node 110, as Figure 15(b) shows. This routing path through node 114 (marked in green Figure 15(b)) is the backup route of node 137. Then the attacking program uses the same method to jam the other four links on the primary routing path and derive the backup routes of each node, as Figure 15(c) to Figure 15(f) show.

Observation 10*: An attacker can derive the primary routing path and backup routes under realistic traffic.*

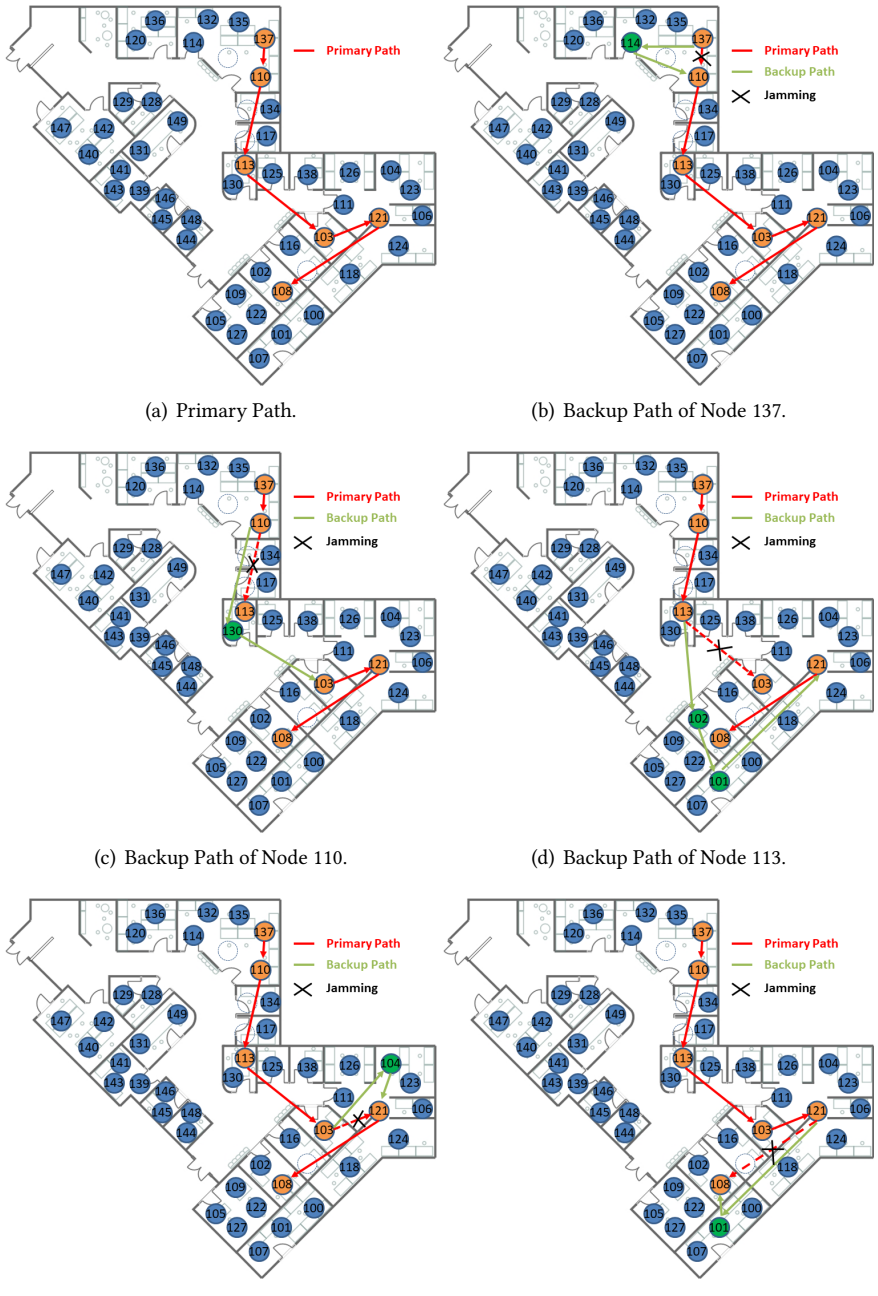
Observation 11*: It takes more time to gather enough observations to crack the routing paths when the transmission failures increase.*

Observation 12*: The time consumption mainly depends on the time consumed to snoop the channels and eavesdrop enough packets.*

6.5 Impact of the Number of Hops

To study the impact of the number of hops on a routing path on the cracking performance, we perform ten sets of experiments and vary the number of hops from 1 to 10. We repeat each set of experiments 20 times. As Table 7 shows, our attacking program always successfully derives the primary routing paths. For instance, the primary routing path from node 100 to node 105 consists of 6 nodes and 5 transmissions, which are scheduled at T, $T+2$, $T+5$, $T+7$, and $T+9$, respectively. Figure 16(a) plots the time consumed to snoop the channels and eavesdrop enough packets. As Figure 16(a) shows, the average snooping time experiences a small increase from 1 slotframe at the sixth set to 1.3 slotframes at the last set. This is because the snooping time of the primary path mainly depends on the retransmission rate. When the retransmission rate increases, the number of hops increases and the routing path becomes more complex. Figure 16(b) shows the execution time to analyze the eavesdropped packets. The cracking time shows an ascending trend from 1.29ms at one hop to 1.43ms at six hops and finally reaches 1.60ms at ten hops. The attacking program has to spend more time to identify the increasing intermediate nodes and confirm the time slots of the transmissions.

Observation 13*: It requires more time for cracking when there are more hops on the routing path.*



(a) Primary Path.

(b) Backup Path of Node 137.

(c) Backup Path of Node 110.

(d) Backup Path of Node 113.

(e) Backup Path of Node 103.

(f) Backup Path of Node 121.

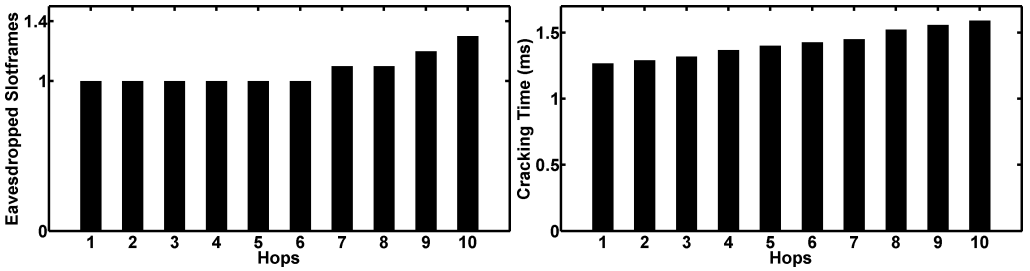
Fig. 15. Backup path cracking process.

6.6 Suggestions to WirelessHART

Here are some suggestions gathered from our case study:

Table 7. Cracking Result

Hops	Routing Path and Time Slot
1	$118 \xrightarrow{T} 121$
2	$100 \xrightarrow{T} 121 \xrightarrow{T+4} 103$
3	$100 \xrightarrow{T} 107 \xrightarrow{T+2} 121 \xrightarrow{T+5} 101$
4	$100 \xrightarrow{T} 107 \xrightarrow{T+2} 121 \xrightarrow{T+5} 101 \xrightarrow{T+7} 102$
5	$100 \xrightarrow{T} 107 \xrightarrow{T+2} 121 \xrightarrow{T+5} 101 \xrightarrow{T+7} 109 \xrightarrow{T+9} 105$
6	$105 \xrightarrow{T} 109 \xrightarrow{T+2} 101 \xrightarrow{T+4} 121 \xrightarrow{T+10} 103 \xrightarrow{T+12} 113 \xrightarrow{T+14} 112$
7	$115 \xrightarrow{T} 113 \xrightarrow{T+2} 103 \xrightarrow{T+4} 121 \xrightarrow{T+10} 103 \xrightarrow{T+12} 113 \xrightarrow{T+14} 110 \xrightarrow{T+16} 114$
8	$120 \xrightarrow{T} 119 \xrightarrow{T+2} 113 \xrightarrow{T+4} 103 \xrightarrow{T+6} 121 \xrightarrow{T+12} 103 \xrightarrow{T+14} 113 \xrightarrow{T+16} 110 \xrightarrow{T+18} 114$
9	$129 \xrightarrow{T} 120 \xrightarrow{T+2} 119 \xrightarrow{T+4} 113 \xrightarrow{T+6} 103 \xrightarrow{T+8} 121 \xrightarrow{T+16} 103 \xrightarrow{T+18} 113 \xrightarrow{T+20} 119 \xrightarrow{T+22} 128$
10	$129 \xrightarrow{T} 120 \xrightarrow{T+2} 119 \xrightarrow{T+4} 113 \xrightarrow{T+6} 103 \xrightarrow{T+8} 121 \xrightarrow{T+16} 103 \xrightarrow{T+18} 113 \xrightarrow{T+20} 110 \xrightarrow{T+22} 114 \xrightarrow{T+24} 136$



(a) Snooping Time. (b) Cracking Time.

Fig. 16. Cracking performance with different hops .

- The specification on using slotframes with 2^n slots in WirelessHART makes the channel hopping sequences easier to be derived. It is beneficial to use a prime number as the slotframe length.
- It is beneficial to use more channels (increasing S_{length}), which increases the cracking difficulty.
- The unencrypted packet headers allow an attacker to crack the graph routes. It is beneficial to encrypt some routing related data fields in the packet header.

7 LESSONS LEARNED

In this section, we provide a series of insights on how to secure the TSCH channel hopping and graph routes based on our analysis and case studies.

7.1 Slotframe Length

As Figure 10(a) shows, the length of combined slotframe makes a significant effect on the cracking difficulty. The larger the combined slotframe is, the more difficult the cracking is. It is beneficial to set the number of slots in different slotframes to be co-prime integers, maximizing the length of combined slotframe. As an example, the individual slotframes in the first three settings (listed in Table 4) have similar lengths, but the cracking difficulty under the third setting is much higher than the others, as Figure 10(a) shows. The default slotframe lengths in Orchestra are prime numbers,

while the slotframes in WirelessHART include 2^n slots, sharing common factors with each other. Comparing Figure 12(a) against Figure 8, cracking the channel hopping sequences in WirelessHART is much easier than cracking Orchestra. Therefore, we would suggest employing multiple slotframes for different types of traffic, and even for different data flows belonging to the same type of traffic, and configuring the number of slots in each slotframe to be a prime number. For instance, if 31, 61, 127, 257, 509, and 1,021 are used as the slotframe lengths for the six data flows in WirelessHART (replacing the setup in Section 6.1), the combined slotframe includes more than 3.20×10^{13} time slots. The attacker has to spend more than 15,256 years to snoop a complete combined slotframe.

7.2 Repetition Pattern

The network device running Orchestra makes scheduling decisions based on its MAC address with a fixed offset in each slotframe, significantly making the channel usage repetition cycle detectable. As Figure 11(a) and Figure 11(b) show, the strong cyclic behavior of packet transmissions significantly reduces the cracking difficulty. As a comparison, there is no strong pattern observed in WirelessHART, which can be used by the attacker to speed up the cracking. Therefore, we would suggest the designer of transmission scheduler avoid strong repetition pattern and randomize the transmissions. For example, Orchestra can employ pseudo-random numbers to randomize the transmission slots in the routing and time synchronization slotframes.

7.3 Channel Diversity

Using more channels not only improves the network performance but also enhances the channel hopping security. As Figure 13(a) shows, the cracking difficulty increases when using more channels. Moreover, the channel hopping sequence used by each device repeats in every $LCM(N_s, S_{length})$ time slots (Section 3.1). Hence, a large S_{length} without any common factor with N_s significantly extends the repetition cycle, making it hard for an attacker to identify the channel repetition pattern. Therefore, we would suggest using all available channels and choose N_s without having a common factor with S_{length}.

7.4 Link Setting

Orchestra specifies that all links associating with the same slotframe uses a single $ChannelOffset$. This design not only limits the network capacity but also significantly reduces the size of channel offset table which is created and maintained by the attacker. Because of the small number of $ChannelOffset$ (up to three), the attacker can perform the cracking very memory-efficiently. Therefore, we would suggest using available $ChannelOffset$ (S_{length}).

7.5 Data Field Encryption

As Table 6 and Figure 15 show, the unencrypted data fields in the packet headers disclose the routing information. An attacker can use such information to derive the graph routes. Figure 14 and Figure 16 show that the attacker can crack the routing paths of WirelessHART networks in a short time bounded by the data period. Hence, it is beneficial to encrypt some data fields in the packet header to enhance the security. Considering the overhead, we would suggest encrypting the original source and final destination addresses located in the NPDU header to increase the difficulty of grouping the eavesdropped packets.

8 RELATED WORK

Jamming attacks have been extensively studied in the WSN and wireless mesh network literature. Simply jamming a channel or the whole spectrum continuously can be easily detected and located by a WIPS [27, 29, 39, 55, 56, 59]. Many countermeasures have been developed in the literature to

minimize the damage. For instance, countermeasure strategies (e.g., adapting frequencies/codes to enforce spread-spectrum techniques) can be implemented in the physical layer to make jamming too complicated to carry out [8, 35, 45, 46, 59, 60]. Adjusting routing [10, 20, 30, 57], adapting transmission power [58], hopping channel [21, 57], adding redundancy [57], increasing randomness on channel access [4, 9, 49] have been shown effective against jamming attacks. Compared to continuous jamming, selective (reactive) jamming has been shown to be much harder to detect [37, 38, 41, 45, 46, 53]. Selective jammers jam wireless channels only when their target devices are transmitting or specific packets of "high" importance are being transmitted, thus making them active for a short period of time and expending orders of magnitude with less energy. Recent studies have shown that the selective jammers can be implemented on inexpensive commercial off-the-shelf (COTS) platforms, making it a realistic threat to wireless communications [37, 38, 41, 53]. However, the existing solutions may fail to distinguish the damage caused by attacks from the normal signal fluctuations, because the transmission failures caused by the attacks happen occasionally and are buried in the normal fluctuations of low-power links. In this paper, we consider a specific kind of selective jamming, tailored to attack TSCH based wireless networks, where jamming is selectively performed against specific communication channels in specific slots.

Although the series of numbers in a TSCH hopping sequence repeats theoretically, it is a challenge to derive the repetition pattern because of skipping scheduled transmissions at runtime. In contrast to the previous studies [50, 62] that only analyze the ideal case and assume S_{length} to be equal to $N_{channel}$, this paper represents the first systematic study that investigates the security vulnerability of TSCH channel hopping in IEEE 802.15.4e under realistic traffic. More important, this paper details a step-by-step attacking approach and presents two case studies with real-world TSCH implementations running on a physical testbed. The experimental results show that an attacker can reverse engineer the channel hopping sequences by silently observing the channel activities, making selective jamming attack a realistic threat to the TSCH networks. There also exist some efforts to derive channel hopping sequences used in Bluetooth piconets [2, 15, 43]. Those methods are not applicable to derive TSCH channel hopping sequences as Bluetooth piconets and TSCH networks use very different schemes to generate channel hopping sequences. In a Bluetooth piconet, the piconet address and the clock of the piconet master are utilized to generate a pseudo random sequence for channel hopping, while TSCH networks use Eq. 1 to generate the channel hopping sequences.

The current industrial WSAN standards (e.g., WirelessHART and ISA100) equip many security features to protect the network against such attacks as denial of service (DoS), MAC spoofing, man in the middle (MITM), and authentication and encryption cracking. There has been an increasing interest in investigating security issues in those standards. For instance, Raza et al. analyzed the potential attacks to WSANs and proposed a series of security enhancement mechanisms for WirelessHART [40]. Alcazar et al. presented a series of vulnerabilities on the routing of WirelessHART and ISA100 and proposed some countermeasures [3]. Pietro et al. developed a distributed self-healing protocol to enhance the intrusion-resilience [32]. Chakrabarty et al. proposed a software defined networking (SDN) architecture to mitigate traffic analysis and data gathering attacks [6]. IEEE 802.15.4 also provides security features such as data confidentiality, data authenticity, and replay protection of MAC frames. For example, the standard includes a security suite based on the AES 128 bits symmetric-key cryptography and supports three different security modes: encryption only (CTR), authentication only (CBC_MAC), and both encryption and authentication (CCM). Unfortunately, the above security features cannot prevent an attacker from cracking the channel hopping sequences by silently observing the channel activities. Our work is therefore orthogonal and complementary.

9 CONCLUSIONS

To meet the stringent real-time and reliability requirements posed by industrial IoT applications, IEEE 802.15.4-based WSANs made a set of unique design choices including employing TSCH and graph routing that distinguish themselves from traditional WSNs that require only best effort services. The function-based channel hopping in TSCH simplifies the network operations at the cost of security. Our study shows that an attacker can reverse engineer the channel hopping sequences and graph routes by silently observing the packet transmission activities, and then launch selective jamming attacks, which are more efficient and stealthy than constant jamming and random jamming. This paper represents the first systematic study that investigates the security vulnerability of TSCH channel hopping and graph routing under realistic settings, demonstrates the cracking process, and presents two case studies using publicly accessible implementations. Finally, this paper provides a set of insights gathered from our analysis and case studies to secure the TSCH channel hopping and graph routing by increasing the cracking difficulty.

For future work, we plan to investigate how an attacker identifies the key messages delivering in the network and reduces the probability of being detected by WIPS when launching selective jamming attacks. Once we fully understand the problem from an attacker's point of view, then we can develop strategies to efficiently detect the selective jamming attacks and defense solutions in future.

ACKNOWLEDGMENT

This work was supported by the NSF through grant CRII-1657275 (NeTS).

REFERENCES

[1] 802.15.4e. 2013. IEEE802.15.4e WPAN Task Group. Retrieved September 28, 2018 from http://www.ieee802.org/15/pub/TG4e.html

[2] Wahhab Albazrqaoe, Jun Huang, and Guoliang Xing. 2016. Practical Bluetooth Traffic Sniffing: Systems and Privacy Implications. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*. ACM, New York, NY, USA, 333–345. https://doi.org/10.1145/2906388.2906403

[3] Cristina Alcaraz and Javier Lopez. 2010. A Security Analysis for Wireless Sensor Mesh Networks in Highly Critical Systems. *IEEE Transactions on Systems, Man, and Cybernetics* 40, 4 (July 2010), 419–428. https://doi.org/10.1109/TSMCC.2010.2045373

[4] Farhana Ashraf, Yih-Chun Hu, and Robin H. Kravets. 2012. Bankrupting the jammer in WSN. In *Proceedings of the 2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS) (MASS '12)*. IEEE, Washington, DC, USA, 317–325. https://doi.org/10.1109/MASS.2012.6502531

[5] Carlo Alberto Boano, Thiemo Voigt, Claro Noda, Kay Römer, and Marco Zuniga. 2011. JamLab: Augmenting sensornet testbeds with realistic and controlled interference generation. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IEEE, 175–186.

[6] Shaibal Chakrabarty, Daniel W. Engels, and Selina Thathapudi. 2015. Black SDN for the Internet of Things. In *Proceedings of the 2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS) (MASS '15)*. IEEE, Washington, DC, USA, 190–198. https://doi.org/10.1109/MASS.2015.100

[7] Xia Cheng, Junyang Shi, and Mo Sha. 2019. Cracking the Channel Hopping Sequences in IEEE 802.15.4e-Based Industrial TSCH Networks. In *Proceedings of the International Conference on Internet of Things Design and Implementation (IoTDI '19)*. ACM, New York, NY, USA, 130–141. https://doi.org/10.1145/3302505.3310075

[8] Jerry T. Chiang and Yih-Chun Hu. 2011. Cross-Layer Jamming Detection and Mitigation in Wireless Broadcast Networks. *IEEE/ACM Transactions on Networking* 19, 1 (Feb. 2011), 286–298. https://doi.org/10.1109/TNET.2010.2068576

[9] Roberta Daidone, Gianluca Dini, and Marco Tiloca. 2014. A Solution to the GTS-based Selective Jamming Attack on IEEE 802.15.4 Networks. *Wireless Networks* 20, 5 (July 2014), 1223–1235. https://doi.org/10.1007/s11276-013-0673-y

[10] Jing Deng, Richard Han, and Shivakant Mishra. 2003. A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks. In *Proceedings of the 2nd international conference on Information processing in sensor networks (IPSN'03)*. Springer-Verlag Berlin, Heidelberg, 349–364. https://doi.org/10.1007/3-540-36978-3_23

[11] Adam Dunkels. 2002. Contiki: The Open Source OS for the Internet of Things. Retrieved September 28, 2018 from http://www.contiki-os.org/

[12] Simon Duquennoy, Atis Elsts, Beshr Al Nahas, and George Oikonomo. 2017. TSCH and 6TiSCH for Contiki: Challenges, Design and Evaluation. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, Piscataway, NJ, USA. https://doi.org/10.1109/DCOSS.2017.29

[13] Simon Duquennoy, Beshr Al Nahas, and Atis Elsts. 2018. 6TiSCH Implementation. Retrieved September 29, 2018 from https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-TSCH-and-6TiSCH

[14] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, New York, NY, USA, 337–350. https://doi.org/10.1145/2809695.2809714

[15] FTE. 2019. Fte comprobe bpa 600. http://www.fte.com/products/BPA600.aspx

[16] Dolvara Gunatilaka, Mo Sha, and Chenyang Lu. 2017. Impacts of Channel Selection on Industrial Wireless Sensor-Actuator Networks. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. IEEE, Piscataway, NJ, USA. https://doi.org/10.1109/INFOCOM.2017.8057049

[17] HART. 2019. HART Communication Protocol and Foundation (Now the FieldComm Group). https://fieldcommgroup.org/

[18] IETF. 2018. IPv6 over the TSCH mode of IEEE 802.15.4e. Retrieved September 28, 2018 from https://datatracker.ietf.org/wg/6tisch/documents/

[19] ISA100. 2018. ISA100. http://www.isa100wci.org/

[20] Chris Karlof, Naveen Sastry, and David Wagner. 2004. TinySec: a Link Layer Security Architecture for Wireless Sensor Networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys '04)*. ACM, New York, NY, USA, 162–175. https://doi.org/10.1145/1031495.1031515

[21] Loukas Lazos, Sisi Liu, and Marwan Krunz. 2009. Mitigating Control-channel Jamming Attacks in Multi-channel Ad Hoc Networks. In *Proceedings of the second ACM conference on Wireless network security (WiSec '09)*. ACM, New York, NY, USA, 169–180. https://doi.org/10.1145/1514274.1514299

[22] Philip Levis. 2013. TinyOS Documentation Wiki. Retrieved September 28, 2018 from http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Documentation_Wiki

[23] Bo Li, Yehan Ma, Tyler Westenbroek, Chengjie Wu, Humberto Gonzalez, and Chenyang Lu. 2016. Wireless Routing and Control: a Cyber-Physical Case Study. In *Proceedings of the 7th International Conference on Cyber-Physical Systems (ICCPS '16)*. IEEE, Piscataway, NJ, USA. https://doi.org/10.1109/ICCPS.2016.7479131

[24] Bo Li, Lanshun Nie, Chengjie Wu, Humberto Gonzalez, and Chenyang Lu. 2015. Incorporating Emergency Alarms in Reliable Wireless Process Control. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems (ICCPS '15)*. ACM, New York, NY, USA, 218–227. https://doi.org/10.1145/2735960.2735983

[25] Bo Li, Zhuoxiong Sun, Kirill Mechitov, Gregory Hackmann, Chenyang Lu, Shirley J. Dyke, Gul Agha, and Billie F. Spencer Jr. 2013. Realistic Case Studies of Wireless Structural Control. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems (ICCPS '13)*. ACM, New York, NY, USA, 179–188. https://doi.org/10.1145/2502524.2502549

[26] Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. 2016. Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems. *Proceedings of the IEEE, Special Issue on Industrial Cyber Physical Systems* 104, 5 (May 2016), 1013–1024. https://doi.org/10.1109/JPROC.2015.2497161

[27] Zhuo Lu, Wenye Wang, and Cliff Wang. 2014. Modeling, Evaluation and Detection of Jamming Attacks in Time-Critical Wireless Applications. *IEEE Transactions on Mobile Computing* 13, 8 (Aug. 2014), 1746–1759. https://doi.org/10.1109/TMC.2013.146

[28] James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. 2013. Disruptive Technologies: Advances that will Transform Life, Business, and the Global Economy. http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/disruptive-technologies

[29] Aristides Mpitziopoulos, Damianos Gavalas, Charalampos Konstantopoulos, and Grammati Pantziou. 2009. A Survey on Jamming Attacks and Countermeasures in WSNs. *IEEE Communications Surveys and Tutorials* 11, 4 (2009), 42–56.

[30] Hossen Mustafa, Xin Zhang, Zhenhua Liu, Wenyuan Xu, and Adrian Perrig. 2012. Jamming-Resilient Multipath Routing. *IEEE Transactions on Dependable and Secure Computing* 9, 6 (Nov. 2012), 852–864. https://doi.org/10.1109/TDSC.2012.69

[31] Raspberry Pi. 2019. Raspberry Pi. https://www.raspberrypi.org/

[32] Roberto Di Pietro, Gabriele Oligeri, Claudio Soriente, and Gene Tsudik. 2010. Intrusion-Resilience in Mobile Unattended WSNs. In *Proceedings of the 29th conference on Information communications (INFOCOM'10)*. IEEE, Piscataway, NJ, USA, 2303–2311. https://doi.org/10.1109/INFCOM.2010.5462056

[33] Kristofer S. J. Pister. 2010. Smart Dust: Autonomous Sensing and Communication in a Cubic Millimeter. https://people.eecs.berkeley.edu/~pister/SmartDust/

[34] Kristofer S. J. Pister and Lance Doherty. 2008. TSMP: Time Synchronized Mesh Protocol. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 391–398.

[35] Christina Popper, Mario Strasser, and Srdjan Capkun. 2010. Anti-jamming Broadcast Communication Using Uncoordinated Spread Spectrum Techniques. *IEEE Journal on Selected Areas in Communications* 28, 5 (June 2010), 703–715. https://doi.org/10.1109/JSAC.2010.100608

[36] Michael E. Porter and James E. Heppelmann. 2014. How Smart, Connected Products are Transforming Competition. https://hbr.org/2014/11/how-smart-connected-products-are-transforming-competition

[37] Alejandro Proaño and Loukas Lazos. 2010. Selective Jamming Attacks in Wireless Networks. In *2010 IEEE International Conference on Communications*. IEEE, Piscataway, NJ, USA, 1–6. https://doi.org/10.1109/ICC.2010.5502322

[38] Alejandro Proaño and Loukas Lazos. 2012. Packet-hiding Methods for Preventing Selective Jamming Attacks. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (Jan. 2012), 101–114. https://doi.org/10.1109/TDSC.2011.41

[39] David R. Raymond and Scott F. Midkiff. 2008. Denial-of-Service in Wireless Sensor Networks: Attacks and Defenses. *IEEE Pervasive Computing* 7, 1 (Jan. 2008), 74–81. https://doi.org/10.1109/MPRV.2008.6

[40] Shahid Raza, Adriaan Slabbert, Thiemo Voigt, and Krister Landernäs. 2009. Security considerations for the WirelessHART protocol. In *Proceedings of the 14th IEEE international conference on Emerging technologies and factory automation (ETFA'09)*. IEEE, Piscataway, NJ, USA, 242–249. https://doi.org/10.1109/ETFA.2009.5347043

[41] Andréa Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. 2013. An Efficient and Fair MAC Protocol Robust to Reactive Interference. *IEEE/ACM Transactions on Networking* 21, 3 (June 2013), 760–771. https://doi.org/10.1109/TNET.2012.2210241

[42] RPL. 2012. RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. Retrieved September 28, 2018 from https://tools.ietf.org/html/rfc6550

[43] Mike Ryan. 2013. Bluetooth: With Low Energy Comes Low Security. In *Presented as part of the 7th USENIX Workshop on Offensive Technologies*. USENIX, Washington, D.C. https://www.usenix.org/conference/woot13/workshop-program/presentation/Ryan

[44] Mo Sha. 2016. Testbed at the State University of New York at Binghamton. Retrieved September 28, 2018 from http://www.cs.binghamton.edu/%7emsha/testbed

[45] Michael Spuhler, Domenico Giustiniano, Vincent Lenders, Matthias Wilhelm, and Jens B. Schmitt. 2014. Detection of Reactive Jamming in DSSS-based Wireless Communications. *IEEE Transactions on Wireless Communications* 13, 3 (March 2014), 1593–1603. https://doi.org/10.1109/TWC.2013.013014.131037

[46] Mario Strasser, Boris Danev, and Srdjan Čapkun. 2010. Detection of Reactive Jamming in Sensor Networks. *ACM Transactions on Sensor Networks* 7, 2 (Aug. 2010), 16:1–16:29. https://doi.org/10.1145/1824766.1824772

[47] TelosB. 2013. TelosB Datasheet provided by MEMSIC. Retrieved October 2, 2018 from http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

[48] Adam Thierer and Andrea Castillo. 2015. Projecting the Growth and Economic Impact of the Internet of Things. https://www.mercatus.org/publication/projecting-growth-and-economic-impact-internet-things

[49] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K. Das. 2017. JAMMY: a Distributed and Self-Adaptive Solution against Selective Jamming Attack in TDMA WSNs. *IEEE Transactions on Dependable and Secure Computing* 14, 4 (July 2017), 392–405. https://doi.org/10.1109/TDSC.2015.2467391

[50] Marco Tiloca, Domenico De Guglielmo, Gianluca Dini, Giuseppe Anastasi, and Sajal K. Das. 2018. DISH: DIstributed SHuffling against Selective Jamming Attack in IEEE 802.15.4e TSCH Networks. *ACM Transactions on Sensor Networks (TOSN)* 15, 1 (Feb. 2018), 3:1–3:28. https://doi.org/10.1145/3241052

[51] Wireless Cyber-Physical Simulator (WCPS). 2018. Wireless Cyber-Physical Simulator (WCPS). Retrieved October 2, 2018 from http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator

[52] Wi-Spy. 2018. Wi-Spy USB Spectrum Analyzer. http://www.wi-spy.co.uk/index.php/products

[53] Matthias Wilhelm, Ivan Martinovic, Jens B. Schmitt, and Vincent Lenders. 2011. Short Paper: Reactive Jamming in Wireless Networks How Realistic is the Threat?. In *Proceedings of the fourth ACM conference on Wireless network security (WiSec '11)*. ACM, New York, NY, USA, 47–52. https://doi.org/10.1145/1998412.1998422

[54] WirelessHART. 2019. WirelessHART. https://fieldcommgroup.org/technologies/hart/hart-technology

[55] Anthony D. Wood and John A. Stankovic. 2002. Denial of Service in Sensor Networks. *Computer* 35, 10 (Oct. 2002), 54–62. https://doi.org/10.1109/MC.2002.1039518

[56] Anthony D. Wood, John A. Stankovic, and S.H. Son. 2003. JAM: a Jammed-area Mapping Service for Sensor Networks. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS '03)*. IEEE, Washington, DC, USA, 286–297. https://doi.org/10.1109/REAL.2003.1253275

[57] Anthony D. Wood, John A. Stankovic, and Gang Zhou. 2007. DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, Piscataway, NJ, USA, 60–69. https://doi.org/10.1109/SAHCN.2007.4292818

[58] Wenyuan Xu, Ke Ma, Wade Trappe, and Yanyong Zhang. 2006. Jamming Sensor Networks: Attack and Defense Strategies. *IEEE Network* 20, 3 (May 2006), 41–47. https://doi.org/10.1109/MNET.2006.1637931

[59] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. 2005. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '05)*. ACM, New York, NY, USA, 46–57. https://doi.org/10.1145/1062689.1062697

[60] Wenyuan Xu, Timothy Wood, Wade Trappe, and Yanyong Zhang. 2004. Channel Surfing and Spatial Retreats: Defenses Against Wireless Denial of Service. In *Proceedings of the 3rd ACM workshop on Wireless security (WiSe '04)*. ACM, New York, NY, USA, 80–89. https://doi.org/10.1145/1023646.1023661

[61] Fan Zhang, Reiner Dojen, and Tom Coffey. 2011. Comparative Performance and Energy Consumption Analysis of Different AES Implementations on a Wireless Sensor Network Node. *International Journal of Sensor Networks* 10, 4 (Oct. 2011), 192–201. https://doi.org/10.1504/IJSNET.2011.042767

[62] Dimitrios Zorbas, Panayiotis Kotzanikolaou, and Christos Douligeris. 2018. R-TSCH: Proactive Jamming Attack Protection for IEEE 802.15.4-TSCH Networks. In *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 00766–00771. https://doi.org/10.1109/ISCC.2018.8538705