Mitigating Stealthy Jamming Attacks in Low-power and Lossy Wireless Networks

Jeongyoon Heo, Jung-Jun Kim, Jeongyeup Paek, and Saewoong Bahk

Abstract: Jamming in wireless networks has advanced to be more stealthy and long-lasting with limited energy. Stealthy attackers transmit short jamming signals to become less detectable with less energy, and yet powerful enough to ruin entire packet transmission procedures. In this article, we deal with three types of stealthy attacks: ‘Reactive jamming’, ‘Jamming ACK’, and ‘Fake ACK’ attacks. These attacks are fatal to low-power and lossy wireless network (LLN) applications because they not only interfere with communication, but also cause LLN devices to quickly drain their batteries. To confront these attacks, we present Dodge-Jam, a lightweight anti-jamming technique suitable for LLN environments to address the stealthy jamming attacks. It protects ACK packet by switching the ACK channel to another channel calculated based on the content of a data packet. Moreover, by partitioning a packet into multiple blocks and performing logical shifts of the blocks when retransmitting the packet, it helps the receiver recover the original packet from received multiple erroneous packets. We implement Dodge-Jam on practical embedded devices, and evaluate its performance through mathematical analysis and experiments on a multi-hop LLN testbed. Our results show that Dodge-Jam successfully avoids many stealthy jamming attacks, recovers jammed packets, and significantly improves packet delivery performance of single-hop and multi-hop networks with small overhead.

Index Terms: Jamming, low-power and lossy network (LLN), IEEE 802.15.4, security, wireless sensor network

1. INTRODUCTION

LOW-POWER and lossy multihop networks (LLNs) have been used widely in many areas including smart grids [2], [3], wireless sensor networks (WSN) [4]–[6], and the Internet of Things (IoT). However, LLNs comprised of resource-constrained battery-operated devices are vulnerable to jamming attacks that cause denial-of-service (DoS) [7]. When a jammer sends jamming signals in order to disrupt communication, the target node under attack may repeatedly fail its transmission attempts and retransmit the packets, which will severely degrade network performance and quickly exhaust the battery of the device. Jamming can be devastating to applications that are sensitive to delay or loss. For example, if a medical network which monitors patients’ physiological data over an LLN [8] is under attack, serious problems may arise because they are directly linked to patients’ life. Home security systems using WSN [9], [10] also have similar problems. An intruder may use jamming attacks to break into a house by disturbing transmissions of alarm messages to the home control center. Jamming attacks can also affect smart grids and smart factories to make misjudgments and cause erroneous operation [11].

LLN devices may be helpless against well-funded powerful wideband jamming attacks [7]. However, if a battery-operated jammer transmits jamming signals continuously at high power, it will drain its energy very fast resulting in a short overall jamming lifetime. Even if the jammer has a sufficient amount of energy to be powerful and long-lasting, it can be easily detected by legitimate defenders [12]. On the other hand, there are more advanced ‘stealthy’ attacks that are less detectable and long-lasting while consuming minimal energy [13]. In these kinds of attacks, attackers only send jamming signals as needed, just enough to disrupt communication. For example, in ‘Reactive jamming’, an attacker sends jamming signals only when it detects on-going packet transmissions. To avoid reactive jamming, power control techniques have been studied [14], [15]. These techniques lower the jamming probability, but cannot recover a packet once it is jammed.

‘Jamming ACK’ attack is also one of such attacks. Once the jammer detects or predicts an ACK transmission, it sends a short jamming signal exclusively to jam the ACK frame in order to make the legitimate sender continuously retransmit the original packet. Furthermore, the work in [16] introduces ‘Fake ACK’ attack, which disturbs communication by sending a jamming signal during a packet transmission, and then sends a fake ACK in order to make the sender believe that the original packet transmission was successful. There have been several prior works that address these attacks [7], [16]–[20], but most of these efforts either require expensive pre-shared secrets, results in limited effect, or incur excessive overhead to resource and power limited LLN devices.

To address these challenges, we design Dodge-Jam, a lightweight anti-jamming technique for LLNs with small overhead. Dodge-Jam confronts attackers that have the same capability level (in terms of processing, memory, transmission power, and battery) with legitimate LLN devices, assuming all radios are half-duplex. It is composed of three main techniques: ‘ACK channel hopping’, ‘Multi-ACK channel hopping’, and ‘Multi-block data shift’.
To address the fake ACK attack, ACK channel hopping changes the channel on which to send ACK frames from the channel on which data packets have been received. There can be a channel-scanning attacker that scans channels to find and jam specific channels through which a legitimate packet is transmitted [7]. ACK channel hopping is free from such an attack because ACK transmission is too short to be found by the attacker using channel scanning. Multi-ACK channel hopping applies a random rendezvous technique [21] to the ACK channel hopping in order to avoid the jamming ACK attack opportunistically. Multi-block data shift is designed to recover jammed and corrupted packets. The sender partitions a packet into several small blocks and adds a CRC to each block. When retransmitting the packet due to transmission failure (upon no ACK reception due to reactive jamming attack, jamming ACK attack, or natural link loss), it performs a logical shift to the packet with the expectations that some blocks have been successfully received and the attacker always jams similar part(s) of the packet. After a few retransmissions, the receiver can recover the original jammed packet from multiple erroneous packets. A key parameter that determines the tradeoff between error recovery capability and transmission overhead in multi-block data shift is the number of blocks used for each packet. We find the optimal number of blocks through both mathematical analysis and testbed experiments.

The contributions of this work are threefold:

- We study three stealthy jamming attacks in LLNs: ‘Reactive jamming’, ‘Jamming ACK’, and ‘Fake ACK’ attacks, and experimentally measure how destructive they are to communication between LLN devices.
- We design ‘Dodge-Jam’, a lightweight anti-jamming technique for LLNs, that combines ‘ACK channel hopping’, ‘Multi-ACK channel hopping’, and ‘Multi-block data shift’ defense mechanisms to defeat the aforementioned attacks. We then show how each of the components in Dodge-Jam contributes to defeating these attacks.
- We implement Dodge-Jam in ContikiOS on TelosB [22] wireless sensor network platform and experimentally evaluate its performance through practical testbed experiments on both single-hop and multi-hop topologies.

Our evaluation shows that Dodge-Jam achieves significantly better packet delivery ratio with small overhead compared to the network without Dodge-Jam. In some cases, it allows the network under attack to continue to operate and communicate, which would otherwise have been useless without Dodge-Jam.

The remainder of this article is structured as follows. Section II discusses related work, and Section III describes three stealthy jamming attacks in LLNs that are energy-efficient and effective. Then, we propose Dodge-Jam to defeat these jamming attacks in Section IV. In Section V, we describe how we implement the attacks, and our defense mechanism, Dodge-Jam. Section VI evaluates both the impact of the attacks and the performance of Dodge-Jam, including overhead costs, through mathematical analyses and testbed experiments. Finally, Section VII concludes the work.

II. RELATED WORK

To defend against the jamming attacks, spread spectrum techniques such as frequency hopping spread spectrum (FHSS) and direct sequence spread spectrum (DSSS) have been widely used [23]. FHSS avoids jamming by changing the frequency according to the pseudo random sequence that the sender and receiver share. However, if this hopping sequence is exposed, a jammer can attack using the same sequence. Even if the jammer does not know the hopping sequence, channel scanning attack is still possible. To share a hopping sequence safely, uncoordinated frequency hopping (UFH) [24] and random rendezvous [21] techniques have been proposed in which the sender and the receiver randomly meet within certain time slots in order to share a hopping sequence. However, these techniques waste a lot of time slots just to share the hopping sequence. DSSS is a technique that reduces the effect of jamming by spreading the signal over a wide frequency band using a shared code. This can also be attacked if the jammer knows the code. We can use a key-based encryption to securely exchange information like the hopping sequence and DSSS code. However, securely sharing and updating the key itself is challenging in dynamic LLN environments where devices frequently join or leave, and communication pairs are continuously changing. We need safe secret sharing techniques such as public key cryptography, certificate and authentication techniques, and techniques for periodically changing keys. These techniques cause large computational and transmission overheads [25].

Two pieces of work that are the closest to ours are Jam-Buster [26] and DEEJAM [7]. These works target jamming attackers with the same capability class as network nodes, and their goal is to increase the cost of jammers and force them to transmit more jamming signals to achieve effective jamming, which will reduce the lifetime and increase the detectability of the jammers.

In DEEJAM [7], the authors first describe the effectiveness and stealthiness of the interrupt jamming attack (and its variants), and how vulnerable an IEEE 802.15.4 based wireless network can be under such energy-efficient jamming attacks. Then they propose four defense mechanisms: frame masking, channel hopping, packet fragmentation, and redundant encoding. Both the attacks and defense mechanisms are evaluated on MicaZ motes to show that their proposal is effective in mitigating such jamming attacks. However, their solutions are based on pre-shared keys, and rely on tight synchronization between legitimate senders and receivers, resulting in high overhead even in the absence of jamming attacks.

In Jam-Buster [26], the authors point out that low resilience and easy differentiability of protocol control messages, as well as high predictability of node wakeup schedules, are what makes LLNs vulnerable to jamming. They propose Jam-Buster to eliminate such vulnerabilities by using multi-block payloads, equal size packets, and randomized wakeup times of network nodes. The idea of using multi-block payloads is similar to one of our schemes. However, it only targets the duty-cycled MAC, increases the packet transmission overhead by making all packet sizes larger even in the absence of jamming, and cannot handle stealthier Jamming ACK or Fake ACK attacks. Furthermore, unlike our work, both Jam-Buster and DEEJAM evaluate their
schemes only on a single-hop network setup without considering multi-hop scenarios.

To handle the Jamming ACK attack, Zhang et al. proposed JACK, which applies a random backoff for sending an ACK [17]. JACK starts with sending an ACK at a randomly chosen time between \([0, (R - 1) \cdot T_{ACK}]\), where \(T_{ACK}\) is the time for the ACK. They found 7.5 as an optimal value for \(R\) which means that JACK takes 7.5 times longer compared to the original ACK sending. To disable JACK, an attacker may simply send a longer jamming signal which only expends a little more energy. Therefore, it has a limited effect with large overhead. To handle Fake ACK attack, the works in [16], [18]–[20] make efforts to authenticate ACK packets, all of which need expensive pre-shared secrets.

The work [12] by Xu et al. focuses on the methods to detect the existence of jamming attacks without necessarily evading them, and the work by Arkin et al. uses jamming to protect, instead of attack, legitimate communication [27]. There are also techniques that can distinguish Zigbee signals from other interfering signals (e.g., WiFi, Bluetooth) on the same frequency band [28]. Finally, the work in [29] surveys various kinds of jamming attacks and countermeasures in WSNs. It provides a review of jamming techniques such as spot, sweep, barrage, and deceptive jamming, while distinguishing types of jammers such as constant, deceptive, random, and reactive jammers. It also classifies existing anti-jamming and jamming detection techniques as proactive, reactive, and mobile agent-based techniques. According to the classification, Dodge-Jam is a proactive countermeasure which handles energy-efficient reactive jamming attacks (i.e., stealthy jamming attacks).

III. ATTACK MODELS

This article studies and tackles three kinds of stealthy and energy-efficient attacks: ‘Reactive jamming’, ‘Jamming ACK’, and ‘Fake ACK’ attacks. This section describes each of these attacks.

A. Reactive Jamming

A reactive jamming attacker jams a packet ‘reactively’ only when it detects an on-going packet transmission to minimize energy consumption and the risk of getting detected while jamming. If a jammer sends jamming signals continuously, not only it will drain its energy quickly, resulting in a short jamming lifetime, but it will also be easily detected. Therefore, the Reactive jamming attacker ensures that it sends a jamming signal just long enough to drop a packet, but at the same time short enough to not easily be detected while minimizing energy consumption.

The attacker can detect the beginning of an on-going packet transmission by sensing a start-of-frame-delimiter (SFD) signal. Fig. 1(a) shows the data frame format of IEEE 802.15.4 [30]. The synchronization header (SHR) which consists of a preamble sequence and the SFD, always precedes a transmission. After a node receives the SFD field, the SFD pin goes active and interrupts the microcontroller to start reading data from the packet buffer. Fig. 2(a) shows the procedures of reactive jamming. When the attacker detects the SFD, it switches its state to not easily be detected while minimizing energy consumption. The attacker can detect the beginning of an on-going packet transmission by sensing a start-of-frame-delimiter (SFD) signal. Fig. 1(a) shows the data frame format of IEEE 802.15.4 [30]. The synchronization header (SHR) which consists of a preamble sequence and the SFD, always precedes a transmission. After a node receives the SFD field, the SFD pin goes active and interrupts the microcontroller to start reading data from the packet buffer. Fig. 2(a) shows the procedures of reactive jamming. When the attacker detects the SFD, it switches its state to receiving mode to transmission mode, which requires a Rx-Tx switching delay of roughly 192 µs on our platform [31]. After the Rx-Tx switching, the jammer starts jamming to interfere with the detected legitimate transmission.

B. Jamming ACK Attack

Jamming an ACK frame is another kind of attack that is energy efficient and not easily detectable. A jamming ACK attacker disturbs communication by sending a jamming signal that collides with an ACK frame. Normally, when a receiver successfully receives a packet, it sends an ACK frame back to the sender. If the sender does not receive the ACK frame for a specific ACK waiting time, it supposes that the transmission has failed. By hampering the ACK, the jamming ACK attacker ruins the two-way handshake and forces the sender to continuously retransmit the entire packet.

Fig. 2(b) shows an example operation of jamming ACK attack. After a Jamming ACK attacker detects a packet transmission by sensing the SFD, it gets its length value which is located right after the SFD in IEEE 802.15.4 frames. From the length value, the jammer calculates the time when the receiver is going to send the ACK, and sends the jamming signal at that exact time to disturb it.

C. Fake ACK Attack

Fake ACK attack is stealthier than the aforementioned attacks. If a packet transmission keeps failing, the sender assumes that the link quality has degraded and thus updates its neighbor table. Then, the routing layer (e.g., RPL [32]) of the sender may change its routing path to send packets on. This recovery
process at the routing layer may successfully avoid the jamming area and send packets on a new routing path.

A fake ACK attacker jams a data packet, and then sends a legitimate-looking fake ACK to make the sender believe that the receiver has successfully received the packet. When a node receives an ACK, it cannot know who sent the ACK because, as shown in the Fig. 1(b), the ACK frame of IEEE 802.15.4 does not include the address of the ACK sender [30]. Therefore, the attacker can fool the sender by making a fake ACK just with the proper sequence number.

Fig. 2(c) shows an operation example of a fake ACK attacker. When the attacker detects the packet transmission, it gets the length value and the sequence number from the beginning of the packet to construct and send ACK properly. Then, it jams the data packet and creates a fake ACK with the corresponding sequence number. Using the length value, the attacker sends the fake ACK frame at an appropriate time (calculated in the same way as in jamming ACK attack) in which the legitimate receiver would send the ACK when it received the packet successfully.

IV. PROPOSED SCHEME: Dodge-Jam

To evade jamming attacks and recover jammed packets from those attacks described in Section III, we present three defense mechanisms. ACK channel hopping addresses fake ACK attack by making sure that a sender is not fooled by fake ACKs, and multi-ACK channel hopping uses multiple channels when sending ACKs to cope with jamming ACK attack. Multi-block data shift helps the receiver recover a jammed packet by reconstructing the original packet from multiple block-shifted retransmissions when some blocks are transmitted in error. Lastly, we propose Dodge-Jam which adaptively combines the three aforementioned schemes to provide a comprehensive defense against the jamming attacks.

A. ACK Channel Hopping

To evade fake ACK attack, we propose ACK channel hopping in which a sender and a receiver together switch the channel on which to exchange ACK dynamically on a per-packet basis. There are $N$ channels in the available channel set $C$ and a channel $c_i \in C$ is the $i$th element of $C$. $c_i$ is selected and used as a channel on which to send an ACK, $C_{ACK}$. $i$ is calculated as,

$$i = \text{rand}(\text{hash}(\text{data})) \mod N,$$

where $\text{rand}()$ is a pseudo random function which is used to generate a channel randomly. We use $\text{hash}()$ function to generate a 16 bits input for the random function based on the input data. In practice, this can be also a $\text{crc16}()$ function for simplicity. The input data of (1) includes both the MAC header and the MAC payload. Since the MAC header includes a sequence number for each message, including the header ensures that the two consecutive packets with the same payload will have different ACK channels.

For ACK channel hopping, only a node who receives the whole packet without error can calculate the correct ACK channel and send ACK properly. Fig. 3(a) illustrates an example of ACK channel hopping. After the sender sends a packet, it calculates the ACK exchange channel from the data it sent and switches the listening channel to receive the ACK. When the receiver receives the packet, it calculates which channel to send the ACK from the received data. Since the attacker sends jamming signals during the packet transmission, it cannot receive the whole packet as is and compute the proper ACK channel. Therefore, with ACK channel hopping, we can defeat fake ACK attack.

B. Multi-ACK Channel Hopping

ACK channel hopping is also effective for preventing jamming ACK attacks if the ACK channel selection procedures (i.e., (1)) are not known to the attacker. In this scenario, the best alternative that the attacker can use is a brute-force method in which it randomly picks a channel to attack. The likelihood of a successful attack reaches only $1/N$. By default, we use 16 channels from channel 11 to 26 of 2.4 GHz IEEE 802.15.4 channels to calculate $C_{ACK}$. However, we can adapt the channel selection to work on channels in other (e.g., 800/900 MHz) IEEE 802.15.4 frequency bands. Furthermore, in some cases, we may need to use fewer channels for hopping for reasons such as overlapping IEEE 802.11 signals, which may increase the probability of success in jamming by the brute-force method. Thus the number of channels and the channels to be used can be adjusted according to the network environment.

If the procedures of ACK channel hopping are known to the jamming ACK attacker who overhears the whole packet, then the attacker can calculate the ACK channel in the same way as a legitimate receiver and successfully jam the ACK. We call this attack ‘Channel hopping jamming ACK attack’. ACK channel hopping also works against this channel hopping jamming ACK attack.
attacker by making it spend more energy in jamming the ACK frames. However, it cannot completely prevent the attack itself.

To deal with the limitations of ACK channel hopping, we propose multi-ACK channel hopping mechanism that avoids channel hopping jamming ACK attack stochastically. When the receiver receives a packet, it obtains \( n \) channels from the received data by repeatedly using (1). That is, it repeats the calculation of (1) until it obtains \( n \) different channels. Then, it sends \( n \) ACKs in a random sequence of the \( n \) channels. The sender waits for the ACK at a channel that is randomly selected from the calculated \( n \) channels. As the attacker cannot know either the ACK channel of the sender nor the channel sequence of the receiver, it can only jam a randomly selected channel among the \( n \) channels. From this procedure, we lower the probability of successful channel hopping jamming ACK attack to \( 1/n \).

Although multi-ACK channel hopping increases the number of ACK transmissions, it can avoid jamming and reduce the number of retransmissions of data packets. As shown in Fig. 1, an ACK frame is usually much smaller than a data packet, which means multi-ACK channel hopping can improve network performance and decrease the energy consumption of both a receiver and a transmitter with less overhead.

Fig. 3(b) shows an example of multi-ACK channel hopping using 2 channels. From a received packet, the receiver computes two ACK channels, \( C_{ACK1} \) and \( C_{ACK2} \), and decides to send ACKs over \( C_{ACK1} \) and \( C_{ACK2} \) in order. The sender decides the channel on which to wait for the ACK as \( C_{ACK2} \). Then the sender receives the ACK on \( C_{ACK2} \) when the receiver sends the second ACK, given that there is no attack on \( C_{ACK2} \). In this example, we avoid jamming ACK attack with a probability of 1/2. From retransmissions, the probability of successfully avoiding jamming increases rapidly. If the receiver continuously uses two channels to send ACKs, the probability of avoiding jamming is

\[
1 - (1/2)^k, \quad \text{where } k \text{ is the number of total transmissions of the packet.}
\]

The likelihood of success rises 87.5% after the 2 retransmissions and 94% after the 3 retransmissions.

\section*{C. Multi-Block Data Shift}

To reconstruct and recover a data packet under reactive jamming or fake ACK attack, we propose multi-block data shift. In this scheme, a sender partitions a packet into multiple small blocks, and adds an extra CRC for each block at the end of that block. When a receiver receives the packet, it first checks the CRC of the entire packet. If the CRC check passes, it sends an ACK back to the sender. This is identical to the normal default operation of IEEE 802.15.4. Otherwise, the receiver checks the CRC of each block, identifies which blocks pass the CRC check, saves only those blocks which pass the CRC check, and waits for the next retransmission. On the sender side, the sender performs logical shift of the blocks every time it needs to retransmit the packet. Since stealthy attackers jam packets only for a short duration due to energy and detectability reasons, there can be some blocks which are not jammed. Also, since a reactive jamming attacker or a fake ACK attacker who jams the data packet sends jamming signals after it senses the SFD, it is highly probable that jamming signals are inserted at a similar position of the data packet for each transmission. Hence, if the sender transmits shifted packets for retransmissions, the block at which the packet is jammed at the previous transmission may not be jammed when it is retransmitted. Based on this intuition, the receiver recovers the jammed packet from several erroneous retransmissions of shifted packets.

Fig. 3(c) shows an example of multi-block data shift which separates a packet into three blocks. Without loss of generality, we give a number starting from 0 to each block according to the position of the block in the original packet. The sender inserts a 1 B CRC at the end of each block. In front of the payload, it adds 1 B flag which is the number of the first block in the packet to notify the order of the blocks. For example, if a flag is 1, the order of blocks is 1, 2, and 0. If the sender does not receive an ACK after a transmission, it performs the logical shift and retransmits the packet until it receives an ACK for the packet.

\section*{D. Dodge-Jam}

Finally, Dodge-Jam combines the three mechanisms described above to handle all the aforementioned attacks. Fig. 4 shows an operation example of Dodge-Jam which partitions a packet into three blocks. First, a sender sends a multi-blocked packet, and the receiver computes and changes the channel to send an ACK on. If there is a fake ACK attacker, we cannot recover the packet by using multi-block data shift alone because the attacker sends a fake ACK which prevents the sender from knowing the transmission failure and retransmitting. Combining multi-block data shift and ACK channel hopping lets the sender successfully recover the packet even when there is a fake ACK attacker. For the first packet transmission, we use only one ACK channel to decrease overhead when the network is not under attack. If the sender does not receive an ACK, it will shift the packet blocks and retransmits the packet. When retransmitting the packet, we apply multi-ACK channel hopping and increase the number of ACK channel by 1 for each retransmission. While doing this, we limit the maximum number of ACK channels to

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4.png}
\caption{An operation example of Dodge-Jam.}
\end{figure}
$M$ to put a bound on the overhead ($M = 3$ in our experiments).

However, there is a challenging issue. If a sent packet is lost, the number of sent packets at the sender and the number of received packets at the receiver may differ. This inconsistency makes the sender and receiver use different sized channel pools. To address the issue, we use the flag field which is used in multi-block data shift in Section IV.C. In multi-block data shift, the flag field is added to notify the order of the blocks. If we increase this value by 1 for each retransmission, the receiver knows both the exact number of sent packets and the first block which is $\text{flag mod \# of blocks}$.

V. IMPLEMENTATION

Each LLN node, for both attackers and legitimate nodes, is a TelosB clone device [22] with an MSP430 microcontroller and a CC2420 radio operating at 2.4 GHz ISM band with 16 channels. We implement the attacks and Dodge-Jam in ContikiOS 2.7, and the transmission power of each node is configured to $-7 \, \text{dBm}$.

A. Implementing the Attackers

Each attacker described in Section III tries to respond quickly to on-going legitimate transmissions to perform its attack. In particular, the reactive jamming and the fake ACK attacker should start sending jamming signals fast enough to corrupt on-going packet transmissions, and all three attackers need some time to prepare for jamming, which includes Rx-Tx switching delay and the time to construct the jamming signal. For this reason, all the attackers terminate their packet reception process as soon as they gain the necessary information to attack the packet transmission.

To achieve this promptness, the attackers change their receive mode configuration in the radio (i.e., CC2420 chip) from a buffered to unbuffered mode. If a node is on the buffered mode, it accesses the packet data after it buffers the data in the RXFIFO buffer. However, on the unbuffered mode, the CC2420 chip of the receiver forwards the data directly to the microcontroller without buffering when a receiver senses a SFD signal. Therefore, the attackers use the unbuffered mode to get the necessary data in real-time and change their status to transmission mode in order to jam the ongoing packet. By default, CC2420 transceiver uses clear channel assessment (CCA) to send a packet only when the channel is idle. To interfere with an ongoing packet, we turn off the default CCA of the attackers. Furthermore, we also turn off the default autoACK feature since it is unnecessary for the jammer.

B. Implementing Dodge-Jam

There are three main components in Dodge-Jam, which are all implemented at an Dodge-Jam-layer between the link layer and the CC2420 chip. First, to implement multi-block data shift, we disable the default CRC check in CC2420. Then, we make the sender partition a packet into multiple blocks, and add a CRC of each block at the end of the block. We also add a 1 B flag to indicate the partitioning sequence of the packet.

To implement ACK channel hopping and multi-ACK channel hopping, we first disable the default autoACK feature and handle the ACK creation and transmission in the software at Dodge-Jam-layer. When a data packet is delivered to Dodge-Jam-layer from the physical layer, the receiver checks the flag value to decide how many channels to send ACKs on, and then selects the ACK channels by using the content of the frame. The sender also chooses the ACK receiving channels in the same way as the receiver. The number of channels that the sender and the receiver compute depends on the number of transmissions. The receiver sends ACKs in a random sequence of the computed channels and the sender randomly picks one of the computed channels to wait for the ACK. After the ACK exchange, they return to the original channel. Fig. 5 depicts the functional components of Dodge-Jam implementation in ContikiOS.

VI. EVALUATION

In this section, we provide and analyze the measurement results for each of the three attacks, and evaluate how Dodge-Jam defeats or diminishes the effect of those attacks. We experiment on both single-hop and multi-hop testbed setups, and compare the results for three different jammer locations in the multi-hop case. We also investigate the effect of various components and parameter settings in Dodge-Jam.

Table 1 lists the shorthand notations for attacks and defense schemes.
For performance evaluation metrics, we use the packet reception ratio (PRR) and the average number of required transmissions per packet (ATX) defined as,

\[ PRR = \frac{\text{number of successfully received unique packets}}{\text{number of sent unique packets}} \]

\[ ATX = \frac{\text{total number of transmissions}}{\text{number of successfully received unique packets}} \]

ATX is used to measure the overhead due to packet retransmissions, and it should ideally converge to the well-known ETX (expected number of transmissions) metric [33] (or vice versa) in the link layer unless the link is completely broken (in which case, PRR is zero and ETX/ATX becomes infinity). The minimum value of ATX is 1 when all transmissions are successful without any retransmission, and a smaller ATX value indicates better transmission performance.

A. Single-Hop Scenario

First, we study the packet delivery performance for a single-hop unicast transmission scenario with a sender, a receiver, and a jammer. The sender and the receiver are placed 2 m away from each other and at a 1 m height from the floor. The jammer is placed 1.5 m away from the sender and receiver at the same height as shown in Fig. 6. In the default setting, the attacker starts jamming as soon as possible (as depicted in Fig. 2) and sends a 9 bytes jamming signal for both data jamming and ACK jamming. We use 3 blocks when performing multi-block data shift which results in a block size of 17 B for our 45 B payload and 6 B network-layer header. For each experiment, the sender sends 100 data packets at an inter-packet interval of 3 s. Tests are repeated five times and the results are averaged. Error bars represent 95% confidence intervals.

A.1 Single-hop Results.

Table 2 presents the effect of each of the three attackers without any defense mechanism in a single-hop scenario. When there is no attacker (N0, the baseline case), both the PRR and ATX are almost 1, which means that there are very few packet losses and retransmissions. When there is a reactive jammer (A1) or a fake ACK attacker (A2), PRR and ATX becomes zero and infinite, respectively, which means that both attackers cause all packet transmissions to fail. In the case of jamming ACK attack (A3), the sender is unaware of the packet reception at the receiver. Thus, the receiver correctly receives most, if not all, of the packets because the jammer only jams the ACK while the sender retransmits the same packet 4 times due to no ACK reception. After sending the fifth transmission including 4 retransmissions, the sender gives up and sends the next packet. For this reason, although the PRR of this case is 1 (from the receiver’s perspective), the sender exhausts the maximum retransmission count which makes ATX become 5.

Fig. 7 shows the PRR and ATX performance of a link with and without Dodge-Jam using white and gray bars. The first thing to notice is that, without Dodge-Jam, PRR is zero when the link is under reactive jamming (A1) or fake ACK attack (A2) (Fig. 7(a)). However, if Dodge-Jam is in action, PRR results are almost 1 in all cases, which proves that all the attacks have been successfully defeated. When the link is under reactive jamming (A1) or fake ACK attack (A2), Dodge-Jam uses the retransmission-based multi-block data shift mechanism in order to recover the jammed packet, which makes ATX approximately 2.

A.2 Contribution of Individual Components.

To further investigate the contribution of individual components of Dodge-Jam, we plot Fig. 8.

In the case of reactive jamming (A1) attack, if we apply multi-block data shift (D1), ATX is approximately 2 while PRR is 1, which means that the receiver recovers jammed packets after a retransmission in most cases. If we apply ACK channel hopping (D2) in addition to multi-block data shift (D1), a combination denoted as D1+D2, it yields almost the same results with D1...
only case, from which it is clear that D1 is powerful enough to defend against reactive jammer (A1).

However, in the case of fake ACK attack (A2), the receiver cannot recover a jammed packet with multi-block data shift (D1) because the sender does not retransmit the packet once it receives the fake ACK from the attacker. When we apply the combination of D1+D2, it can avoid the fake ACK attack by changing the channel on which to send the ACK. ATX in this case is about 2.4, which is slightly larger than the ATX results under reactive jamming (A1). This is because its jamming position has to be slightly delayed compared to that of a reactive jamming attacker, since a fake ACK attacker listens to the sequence number. Said differently, the reactive jamming attacker would send out jamming signals as soon as possible to jam even the smallest packets, but the fake ACK attacker waits a little longer. This sometimes results in the fake ACK attacker corrupting more than one block (under our block size configuration, which we discuss more later) and increases ATX.

In the case of jamming ACK attack (A3), it is challenging to avoid the attack only with multi-block data shift (D1) because the attacker jams ACK frames in order to make the sender continuously retransmit the packets. In order to avoid A3, the combination of D1+D2 is required to hide the ACK channel from the attacker. This has been successful as can be seen from the fact that ATX is approximately 1.

We also consider another kind of jamming ACK attack which changes the ACK channel in the same way as multi-ACK channel hopping. We call this attack ‘Channel hopping jamming ACK attack’ (A3∗). Fig. 9 shows the experiment results under A3∗. ATX of D1+D2 is 5 which implies that ACK channel hopping (D2) is insufficient to avoid A3∗. The expected ATX of multi-ACK channel hopping (D3), $E[ATX_{D3}]$, is calculated as,

$$E[ATX_{D3}] = \sum_{k=1}^{5} k \cdot \left( \frac{1}{7} \right)^k + 5 \cdot \left( \frac{1}{7} \right)^5,$$

where $n$ is the number of ACK channels used in D3. If D3 uses two channels to send ACK ($n = 2$), $E[ATX_{D3}]$ is approximately 1.9.

In the case of Dodge-Jam, we obtain the expected ATX, $E[ATX_{Dodge-Jam}]$ when M is 3, as

$$E[ATX_{Dodge-Jam}] = I + \sum_{k=3}^{5} k \cdot \left( \frac{1}{3} \right)^{k-2} + 5 \cdot \left( \frac{1}{3} \right)^3,$$

which becomes approximately 2.7. The experimental results in only case, from which it is clear that D1 is powerful enough to defend against reactive jammer (A1).

However, in the case of fake ACK attack (A2), the receiver cannot recover a jammed packet with multi-block data shift (D1) because the sender does not retransmit the packet once it receives the fake ACK from the attacker. When we apply the combination of D1+D2, it can avoid the fake ACK attack by changing the channel on which to send the ACK. ATX in this case is about 2.4, which is slightly larger than the ATX results under reactive jamming (A1). This is because its jamming position has to be slightly delayed compared to that of a reactive jamming attacker, since a fake ACK attacker listens to the sequence number. Said differently, the reactive jamming attacker would send out jamming signals as soon as possible to jam even the smallest packets, but the fake ACK attacker waits a little longer. This sometimes results in the fake ACK attacker corrupting more than one block (under our block size configuration, which we discuss more later) and increases ATX.

In the case of jamming ACK attack (A3), it is challenging to avoid the attack only with multi-block data shift (D1) because the attacker jams ACK frames in order to make the sender continuously retransmit the packets. In order to avoid A3, the combination of D1+D2 is required to hide the ACK channel from the attacker. This has been successful as can be seen from the fact that ATX is approximately 1.

We also consider another kind of jamming ACK attack which changes the ACK channel in the same way as multi-ACK channel hopping. We call this attack ‘Channel hopping jamming ACK attack’ (A3∗). Fig. 9 shows the experiment results under A3∗. ATX of D1+D2 is 5 which implies that ACK channel hopping (D2) is insufficient to avoid A3∗. The expected ATX of multi-ACK channel hopping (D3), $E[ATX_{D3}]$, is calculated as,

$$E[ATX_{D3}] = \sum_{k=1}^{5} k \cdot \left( \frac{1}{7} \right)^k + 5 \cdot \left( \frac{1}{7} \right)^5,$$

where $n$ is the number of ACK channels used in D3. If D3 uses two channels to send ACK ($n = 2$), $E[ATX_{D3}]$ is approximately 1.9.

In the case of Dodge-Jam, we obtain the expected ATX, $E[ATX_{Dodge-Jam}]$ when M is 3, as

$$E[ATX_{Dodge-Jam}] = I + \sum_{k=3}^{5} k \cdot \left( \frac{1}{3} \right)^{k-2} + 5 \cdot \left( \frac{1}{3} \right)^3,$$

which becomes approximately 2.7. The experimental results in

Fig. 9. ATX results under A3∗ with various combination of defense mechanisms. Since this attack only jams the ACK frames and does not jam the data packet, PRR is 1 for all cases from the receiver’s point of view.

To investigate the impact of data shifting and the number of blocks $B$ of multi-block data shift (D1) on Dodge-Jam’s performance, we analyze the expected ATX according to $B$, $E[ATX_B]$, and find the optimal $B$ which minimizes $E[ATX_B]$. Fig. 10 shows two packets with different $B$ when D1 is applied (i.e., $B$ is 2 or 3). Since the attacker starts jamming only after it receives the necessary information from the header, it cannot jam some of the leading part of the data. $\alpha$ in Fig. 10 represents this jam-free part. $l$ is the length of the data in a packet and $j$ is the length of a jamming signal. The unit of each variable is byte. The jamming start position may vary depending on the characteristics of the jammer (e.g., how fast it makes a transition from RX mode to TX mode and starts jamming). Thus, we assume that the starting points of jammers are uniformly distributed between $\alpha$ and $l + B$ ($B$ B are added due to the extra per-block CRCs), and a jammer always jams the same part of a packet.

For simplicity, first we assume that $j$ is smaller than a block. Then, the jamming signal simultaneously jams 1 or 2 blocks in a packet. When the jamming signal spans two blocks, the attacker jams both blocks. The probability that two blocks are jammed by one jamming signal is $\frac{B-1}{B-\alpha}$ Since we need at least two transmissions when a block is jammed and three transmissions when two blocks are jammed, we obtain $E[ATX_B]$ as,

$$E[ATX_B] = \frac{(B-1)j}{B-\alpha} + \left(1 - \frac{(B-1)j}{B-\alpha}\right) \times 2.$$

$B$ should be larger than 2 because, if $B$ is 1 or 2, then there are cases where all the blocks are jammed and none can be recovered, which makes the ATX value infinite.

To find the optimal $B$, we obtain the difference between the probability of jamming two blocks when $B = b + 1$ and $b$ as,

$$\frac{(b)j}{l + (b + 1) - \alpha} - \frac{(b-1)j}{l + b - \alpha} = \frac{j(l + j - j\alpha)}{(l + b - \alpha)(l + b + 1 - \alpha)}.$$

To succeed in jamming, $l$ should be larger than $\alpha$ (i.e., $l > \alpha$) and then, (7) is greater than 0. This indicates that a larger $B$ exhibits worse ATX performance. Therefore, the optimal $B$ is 3 when $j$, the length of a jamming signal, is smaller than a block. In general, the optimal $B$ is the smallest value that the jammer cannot jam all the blocks using a single jamming signal of length $j$. This implies that it is possible for a jammer
to increase the length of the jamming signal, \( j \), to defeat Dodge-Jam if \( B \) and \( l \) are known. To counter that, Dodge-Jam can adjust \( B \) dynamically if it can detect \( j \). Dynamic adaptation of \( B \) in Dodge-Jam is left as future work.

Fig. 11 plots the experimental results for a varying number of blocks \( B \) when the link is under reactive jamming (A1), and it fits well with our analysis. In this figure, the light gray bars represent the results when multi-block data shift (D1) is applied, and the white bars represent the results when multi-block was used without shifting (to simulate the same packet length and overhead without shifting). Since we do not know the actual characteristics of the attacker in practice, we have implemented ‘Random A1’ attackers with various jamming starting points for this experiment as in the analysis. We average the results of 11 attackers with 11 different starting points. Starting points are uniformly distributed between \( \alpha \) and \( l + B \). \( \alpha \) is the average of the experimental values when jammers start jamming as soon as possible.

ATX is infinite and PRR is 0 for all cases exclusively under multi-block because the receiver cannot recover the jammed packet even after maximum retransmissions. When D1 is applied, the graph shows the lowest ATX value and the best PRR when the number of blocks is 3. When the number of blocks is 2 or 4, sometimes two blocks are jammed depending on the attack position. It decreases the PRR of D1 with 2 blocks, and increases the ATX for the both blocks. If we separate a packet into too few blocks and too many blocks are used because the jamming signal is long enough to jam both blocks. When the length of the jamming signal is 35 B, the attacker even jams 3 blocks and decreases the PRR results of 2 and 3 block cases to 0. Thus, it is possible to recover from a long-length jamming by dividing a block into many smaller blocks, but when the jamming length is short, the probability that several blocks will be broken increases and performance decreases.

An attacker can change its jamming length and starting point to improve its unpredictability. Considering this, we have implemented another advanced ‘Random A1*’ attacker which use 5 different starting points and 3 different jamming length. Note that, in the other results of Fig. 12, the attacker starts jamming as soon as possible. Starting points are uniformly distributed between \( \alpha \) and \( l + B \) and the jamming length has a value between 10 B and 30 B. The ‘random’ in the x-axis of Fig. 12 represents the result under ‘Random A1*’ attack. The result shows the highest PRR and the lowest ATX value when 3 blocks are used. If an attacker uses a long jamming signal and starts jamming transmission late, it only jams the end part of the data packet. Since Dodge-Jam uses ACK channel hopping, the attacker can not interfere with ACK frame transmission, and eventually becomes analogous to sending a short jamming signal. For this reason, the best performance on average is achieved when 3 blocks are used.

As it is hard to know attacker’s operation in advance, using 3 blocks can be a good solution according to the results of VI.A.3.

B. Multi-Hop Scenario

Studying jamming attacks and defense techniques under multi-hop scenarios are important because the LLN routing protocol (e.g., RPL [32] and its variants [34]–[37]) may (or may not) act independently to avoid links that are problematic depending on the attack type. For some attacks, upon detection of insufficient packet delivery performance (e.g., ETX), the routing protocol may naturally select an alternate link for its routing path without any need for jamming defense. Of course, this as-

![Fig. 13. Testbed topology map for the multi-hop scenario.](image-url)
sumes that an alternate link exists. On the other hand, for some other attacks, the routing protocol may not be able to detect link failure and use the same path for an extended amount of time unless some higher-layer action is performed. Our evaluation in this subsection studies these cases.

To study the impact of jamming attacks on the performance of Dodge-Jam in multi-hop network scenarios, we deployed a LLN testbed as depicted in Fig. 13. There are 8 legitimate LLN sender nodes (depicted in circles) and one sink node (marked with a star) in an indoor environment. Solid arrows depict a snapshot of the routing topology during one of our baseline experiments, which shows a 3-hop network. This routing topology is the outcome of RPL, the IETF standard IPv6 routing protocol for low-power and lossy networks [32], with the default MRHOF objective function. At the beginning of each experiment, we let RPL construct the routing topology for 5 min. Once the routing paths have been established, each legitimate LLN node sends 100 end-to-end data packets to the root with an inter-packet interval of 30 s.

Using this testbed setup, we experiment with one attacker at a time at three different locations for the attacker. The first location (attacker 1) is close to the root, disrupting communications between the root and two first-hop nodes which forward all the traffic from their children nodes. For this reason, we expect this attacker location to have the most significant impact on network performance. The second location (attacker 2) is located in the middle of the routing tree and the network. It is expected to affect several links near it. The last location (attacker 3) is farther away from the root, closer to a leaf node which has the highest hop-distance to the root, and interferes with links that are far away from the root. This location is expected to have only limited impact on the performance. The dotted arrows in Fig. 13 represent alternate links that RPL nodes may select as their routing path towards the root if they detect link failure.

Fig. 14 depicts the end-to-end PRR and network-wide ATX performance of the multi-hop RPL network with and without Dodge-Jam. When calculating the network-wide ATX, we take into account all transmissions including retransmissions and forwardings, and divide it by the total number of successfully received packets at the root. This network-wide ATX indirectly shows how much extra effort, on average, was needed by the network to deliver one end-to-end data packet to the root.

The leftmost white bars demonstrate the results without any attack, and shows that the PRR and the ATX of both baseline (N0) and Dodge-Jam are near 1 and 2.5, respectively without any attack. Attacker location 1 attacks both incoming links at the root. Since all the packets must be delivered through these two links, if the attacker acts as a reactive jamming (A1) or fake ACK (A2) attacker, it affects the network (without Dodge-Jam) significantly and decreases the PRR to almost zero, a complete black-out. The ATX is drastically increased under all kinds of attacks for this attacker location without Dodge-Jam.

In the case of attacker locations 2 and 3, there are alternate links on which to send packets. Thus, if the attackers act as a reactive jammer (A1) or jamming ACK attacker (A3), nodes will experience some packet losses, and will change their routing path to send packets on as a natural parent selection process of the routing protocol (RPL in our case). With Dodge-Jam, as it recovers jammed packets under A1 or A2 while not changing the routing path, the ATX results increase slightly but Dodge-Jam increases the PRRs to near 1, which means that it rarely loses packets.

However, nodes are unable to detect link failures under fake ACK attack (A2) without Dodge-Jam. Thus, nodes will repeatedly attempt to transmit packets through jammed links. As can be seen in Fig. 14(a), the PRR results are above 98%, even for those cases with 0% PRR without Dodge-Jam. For the case when the network is under A3, ATX results have decreased with Dodge-Jam in all cases compared to those without Dodge-Jam.

Our experimental results show that Dodge-Jam successfully defeats all three attacks regardless of the attacker location, and allows the network under attack to continue to operate and communicate, which would otherwise have been useless without Dodge-Jam.

C. Overhead

Although Dodge-Jam successfully defeats three stealthy attacks and allows the network to continue to operate efficiently in terms of PRR and ATX, this comes at a cost. Extra compu-
tation, channel switching, and block CRC all contribute to the extra overhead needed for Dodge-Jam, even in the absence of any attack. Thus here we analyze the overhead to adopt Dodge-Jam in terms of additional time delay and bytes transmitted.

Adding block CRCs and the flag are the extra byte overhead for applying multi-block data shift. For example, if we partition a packet into three blocks as discussed in Section VI.A.3, this overhead takes up about 3% of the packet when the size of the packet is 127 B. Instead, if we decide to partition a packet into 5 blocks for better protection, the overhead becomes about 4.7%.

To analyze the time overhead when using Dodge-Jam, we empirically measured the average time needed by a transmitter to send a packet and successfully receive an ACK in the single-hop topology of Fig. 6 under no attack. Fig. 15 shows the result on TelosB mote with ContikiOS. The light gray bar represents the result when using hardware ACK (ACK processed within CC2420 chipset) without any modification, the dark gray bar shows the result when using software ACK (ACK processed in the link layer software stack), and the white bar is for the case of Dodge-Jam. Using software ACK requires approximately 28% more time to send a packet and receive an ACK than using hardware ACK, and using Dodge-Jam requires about 25% more time compared to the case of software ACK.

The reason for the added time overhead is mostly due to the processing delay before sending an ACK, which increases ACK wait time2. Note that, as mentioned in Section V.B, we disabled the default hardware ACK in CC2420 and adopted software ACK method to implement Dodge-Jam. Using software ACK itself incurs software delay before sending an ACK. In addition to this, Dodge-Jam sends an ACK only after re-ordering packet blocks, calculating an appropriate ACK channel, and switching to that ACK channel. These cause extra delay before sending an ACK. Therefore, Dodge-Jam requires longer ACK wait time to receive ACKs properly. Such added delay incurs increase in energy consumption of network devices when duty-cycling MAC protocols (e.g., [31], [38]) are used for low-power operation.

When Dodge-Jam is applied, LLN devices need to stay awake for ~4 more milliseconds to send a packet. However, if we assume that LLN devices usually do not require high throughput and often send packets infrequently, the overhead could be insignificant. For example, if a sender sends one packet every second, the extra time overhead will occupy only ~ 0.4% of a 1 s period. Since Dodge-Jam recovers PRR significantly under the jamming attacks, we believe that additional bytes and time overhead are acceptable.

VII. CONCLUSION

In this article, we have presented an anti-jamming technique, termed Dodge-Jam, for low-power and lossy networks. It is composed of ACK channel hopping, Multi-ACK channel hopping, and Multi-block data shift techniques to evade stealthy jamming attackers and recover packets from jammed transmissions. We implement Dodge-Jam on real low-power embedded devices, and evaluate its performance in both single-hop and multi-hop wireless testbed. Our evaluation results show that Dodge-Jam successfully defeats three types of stealthy and energy-efficient jamming attackers, and recovers packet reception ratio from 0% to over 98% with small overhead in both single-hop and multi-hop scenarios. As our future work, we plan to study how Dodge-Jam defeats other types of attackers, analyze the impact and performance of attackers and Dodge-Jam in terms of energy consumption, and develop techniques to find the location of attackers with the information obtained from the network.

REFERENCES


Jeongyoon Heo is currently a Ph.D. student at the School of Computer Science and Electrical Engineering, Seoul National University, Seoul, Republic of Korea. She received her B.S. degree in Electrical Engineering from Korea University in 2013. Her research interests are in the area of security and energy efficiency in wireless networks, Internet of Things and also in the area of vehicle to everything (V2X) including localization and security.

Jung-Jun Kim received his B.S. degree in Electrical and Computer Engineering from Seoul National University in 2017. He is currently working toward the master degree in Electrical and Computer Engineering, Seoul National University. His research interests include IoT connectivity.

Jeongyeup Paek received his B.S. degree from Seoul National University in 2003 and his M.S. degree from University of Southern California in 2005, both in Electrical Engineering. He then received his Ph.D. degree in Computer Science from the University of Southern California (USC) in 2010 where he was a Member of the Networked Systems Laboratory (NSL) led by Dr. Ramesh Govindan. He worked at Deutsche Telekom Inc. R&D Labs USA as a Research Intern in 2010, and then joined Cisco Systems Inc. in 2011 where he was a Technical Leader in the Internet of Things Group (IoTG), Connected Energy Networks Business Unit (CENBU, formerly the Smart Grid BU). At Cisco, he was one of the Lead Engineers for CG-Mesh system and WPAN software. In 2014, he was with the Hongik University, Department of Computer Information Communication as an Assistant Professor. Jeongyeup Paek is currently an Assistant Professor at Chung-Ang University, School of Computer Science and Engineering, Seoul, Republic of Korea since 2015. He is an IEEE and ACM Member.

Saewoong Bahk is a professor at Seoul National University (SNU). He served as Director of the Institute of New Media and Communications during 2009-2011. Prior to joining SNU, he was with AT&T Bell Laboratories as a Member of Technical Staff from 1991 to 1994 where he had worked on network management. He received KICS Haedong Scholar Award in 2012. He has been leading many industrial projects on 3G/4G/5G and IoT connectivity supported by Samsung Electronics, LG Electronics, SK Telecom, etc and published more than 200 technical papers and holds 72 patents. He is serving as Chief Information Officer (CIO) of SNU, General Chair of IEEE Dynamic Spectrum Access and Networks (DySPAN) 2018, general chair of IEEE Wireless Communication and Networking Conference (WCNC) 2020, and Director of Asia-Pacific region of IEEE ComSoc. He was Vice President of the Korean Institute of Communications and Information Sciences (KICS), TPC Chair for IEEE VTC-Spring 2014, General Chair of JCCI 2015. He is co-EIC of the Journal of Communications and Networks (JCN) and an Editor of IEEE Network Magazine. He was on the Editorial Board of Computer Networks Journal (COMNET) and IEEE Transactions on Wireless Communications (TWireless). He is a Member of the National Academy of Engineering of Korea (NAEKF), an IEEE Senior Member, and a Member of Whos Who Professional in Science and Engineering. He received B.S. and M.S. degrees in Electrical Engineering from SNU in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania in 1991.