BitTrickle: Defending against Broadband and High-power Reactive Jamming Attacks

Yao Liu, Peng Ning
North Carolina State University, Raleigh, NC 27695
{yliu20, pning}@ncsu.edu

***Abstract*—Reactive jamming is not only cost effective, but also hard to track and remove due to its intermittent jamming behaviors. Frequency Hopping Spread Spectrum (FHSS) and Direct Sequence Spread Spectrum (DSSS) have been widely used to defend against jamming attacks. However, both will fail if the jammer jams all frequency channels or has high transmit power. In this paper, we propose BitTrickle, an anti-jamming wireless communication scheme that allows communication in the presence of a broadband and high power reactive jammer by exploiting the reaction time of the jammer. We develop a prototype of BitTrickle using the USRP platform running GNURadio. Our evaluation shows that when under powerful reactive jamming, BitTrickle still maintains communication, whereas other schemes such as 802.11 DSSS fail completely.**

I. Introduction

A reactive jammer stays quiet when a target sender is not transmitting, but jams the channel when it detects transmission from the sender. Compared with constant jamming, reactive jamming is not only cost effective for the jammer, but also hard to track and remove due to its intermittent jamming behaviors [2]. Reactive jamming has been widely used in military applications to cut off the wireless communication of the enemy army or disable radio-controlled devices [2].

Frequency Hopping Spread Spectrum (FHSS) (e.g., [8], [17]) and Direct Sequence Spread Spectrum (DSSS) (e.g., [12], [15], [20]) are dominantly used for anti-jamming purposes. In FHSS, the sender and the receiver switch their communication channel periodically to avoid jamming. In DSSS, the sender multiplies the original message with a pseudo-random sequence to obtain spreading gain. If the jammer's power is not strong enough to overwhelm the DSSS signals with spreading gain, the receiver can use the same pseudo-random sequence to recover the message. However, FHSS, DSSS and their variants all share a common assumption that the jammer can only jam part of channels or has limited transmit power. Unfortunately, if the jammer jams all channels simultaneously or transmits with high power to overcome the spreading gain, these methods fail to maintain communication.

It may appear that a broadband, high-power reactive jammer is perfect and invincible. First, the jammer can jam all channels and overcome the spreading gain. In addition, the reactive strategy arms the jammer with stealthiness, enabling them to avoid detection and removal.

In this paper, we develop BitTrickle, a communication scheme that allows wireless devices to exchange information under broadband, high-power reactive jamming attacks. BitTrickle requires no special hardware. Even wireless devices that are not equipped with spread spectrum capability can use BitTrickle to combat reactive jamming attacks.

BitTrickle achieves the anti-jamming capability by harnessing a subtle opportunity arising from an intrinsic feature of reactive jamming, i.e., "the jammer stays quiet when the channel is idle, but starts transmitting a radio signal as soon as it senses activity on the channel" [19].

Channel sensing is indispensable for a reactive jammer to determine if a target sender is transmitting. Channel sensing causes a short delay. For example, energy detection, the most popular channel sensing approach with very small sensing time [9], requires more than $1ms$ to detect the existence of target signals for a 0.6 detection probability and -110dBm signal strength, when implemented in a fully parallel pipelined FPGA for fast speed [5]. In addition, upon detecting the target signal, the jammer needs to switch its status from quiet to transmitting. The switching process further takes time. As another example, German SGS 2000 series military jammer has a switching time of about $50\mu s$ [3]. Therefore, before the jammer actually jams, the sender has already transmitted ΔtR bits, where Δt is the reaction time of the jammer and R is the transmission rate of the sender.

BitTrickle exploits such unjammed bits to establish jamming-resilient communications. The receiver collects bits that are transmitted by the sender but not jammed by the reactive jammer, and assembles them to construct the original message. It is worth pointing out that BitTrickle also works in defending against random jammers that sleep after jamming for a random time and resume afterwards.

Two technical challenges are addressed in developping BitTrickle. First, the receiver needs to extract unjammed bits from received bit stream. We develop a novel technique that utilizes modulation properties to identify unjammed bits. In addition, an error recovery mechanism is required to tolerate synchronization errors (e.g., lost bits), deal with pollution attacks, and guarantee the performance of error correction codes (ECC). Accordingly, we develop an encoding and decoding technique to enable error recovery with high efficiency.

The throughput of BitTrickle partially depends on the reactive jamming pattern. Long jamming duration lowers the throughput. This is similar to the situation that FHSS and DSSS have a lower bit rate in the presence of a stronger jammer who covers more channels and uses higher power. However, at the same time, the reactive jammer risks higher probability of being detected and removed. The goal of BitTrickle is to raise wireless communication from non-existence

in extremely hostile environments (e.g., battlefield) to being available, rather than support high-speed applications like video streaming in benign environments. When FHSS and DSSS cannot deliver a single bit, BitTrickle can still maintain wireless communication.

A very important application of anti-jamming techniques is tactical communication. FHSS and DSSS systems typically support a bit rate from dozens bps to several kilo bps [3]. For example, PRC 3100H radio systems, which are FHSS transceivers used by US Army to provide encrypted voice and data communication in battlefields, can achieve a bit rate up to 2400bps [3]. Our prototype implementation of BitTrickle on the Universal Software Radio Peripheral (USRP) platform [13] achieves a similar bit rate between 200-2,500 bps. Further, BitTrickle can also be implemented on more advanced hardware (e.g., commercial or military chips) than USRPs to guarantee a desired bit rate in the presence of an extreme jammer with very short reaction time.

The contribution of this paper is three-fold: (1) we develop BitTrickle by exploiting a jammer's reaction time, which enables wireless communication even when previous anti-jamming techniques fail; (2) we develop two novel techniques in BitTrickle, including a modulation error based method to extract unjammed bits from received bit stream, and an encoding and decoding method to recover the original message from message fragments whose positions in the original message are unknown; (3) we implement a BitTrickle prototype using the USRP platform [13], and evaluate the performance of BitTrickle in terms of packet delivery rate and throughput.

The rest of the paper is organized as follows. Section II shows the overview of BitTrickle and two technical challenges we face in this work. Sections III and IV present our methods for addressing these technical challenges. Section V describes the implementation and evaluation. Section VI discusses related work, and Section VII concludes this paper.

II. Overview of BitTrickle

As discussed earlier, BitTrickle exploits the sensing delay of reactive jamming to enable message transmission. In this section, we describe the high-level behaviors of the sender and the receiver, respectively, and then discuss the technical problems that need to be solved to build BitTrickle.

A. Transmission at the Sender

The sender encodes a message using a *BitTrickle encoder*, which enables the receiver to recover the message in the presence of partial message corruption. A message is encoded in a way such that the receiver can identify the boundary of a received encoded message, and thus no extra synchronization preamble is required for transmitting the encoded message. If a message is too long, the sender may first split it into short messages, and then encode those short messages.

To transmit encoded message, the sender needs to find times when the jammer is not jamming. The sender may take a random backoff before each transmission, as shown in Figure 1. This makes it hard for the reactive jammer to predict when the sender will start the next transmission. The

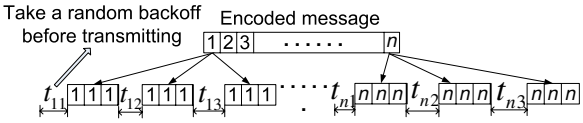

Fig. 1. Transmission at the sender

jammer may attempt to jam the communication for longer time periods. However, this will increase the chance for the reactive jammer to be detected and removed. If the sender resides in the power range of the jammer, the need of random backoffs can be removed. Before each transmission, the sender may perform channel sensing to determine whether or not the jammer is transmitting. If not, the sender immediately sends bits without waiting for the backoff time to expire. The sender may transmit each bit of the encoded message for multiple times to increase the chance that the receiver receives this bit.

Note that BitTrickle can also address random jamming attacks. In both reactive and random jamming scenarios, a common feature is that multiple transmitted bits are lost. In Section IV, we will show how BitTickle deals with lost bits.

Figure 1 shows the most conservative situation, where the sender transmits one bit a time. The sender can improve the performance by transmitting multiple bits a time. To this end, the sender may learn how many bits can be delivered within the jammer's reaction time through, for example, detecting the point where jamming happens or using ACK packets from the receiver, which can be transmitted in a similar way. This paper focuses on the transmission of single bits. Extending the approach to transmitting multiple bits a time is straightforward.

B. Reception at the Receiver

The receiver's task is to extract the unjammed bits and reconstruct the original message from these unjammed bits, which are possibly collected from multiple transmissions.

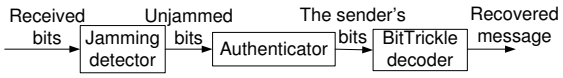

Fig. 2. Reception at the Receiver

Extracting Unjammed Bits: Figure 2 shows the high-level view of the receiver's operations. The receiver processes each received bit with the *jamming detector*, which checks if this bit is jammed, and discard all jammed bits. The output of the jamming detector is thus a collection of unjammed bits.

Dealing with Pollution Attacks: An intelligent jammer may attempt to pollute the unjammed bits to defeat our scheme. Specifically, the jammer transmits fake bits to the receiver when the sender is not transmitting. Those fake bits can cause a high decoding complexity (e.g., exponential complexity) at the receiver. Therefore, the receiver should have the ability to remove the jammer's bits. Accordingly, the receiver feeds the output of the jamming detector to an *authenticator*, which distinguishes the sender's bits from the jammer's bits by using physical layer authentication approaches such as radiometrics (e.g., [4]) and radio frequency (RF) fingerprints (e.g., [14]). As cryptographic authentication

faces cryptanalysis based attacks (e.g., birthday attacks against digital signatures), physical layer authentication may also face similar threats as revealed by [7]. A hybrid of multiple physical layer approaches may be explored to defense against sophisticated attacks. How to improve the authentication capability, including cryptographic and physical layer authentication, is complementary to this work.

We assume that the jammer cannot break physical layer authentication approaches. However, In practice, false negatives and false positives may happen with small probabilities when those approaches are employed. With a false negative, the jammer's bits are identified as the sender's bits. With a false positive, the sender's bits are identified as the jammer's bits. Although false negatives and false positives are events of small probabilities, they may introduce a small number of inserted bits and lost bits. In Section IV, we will show how the receiver handles inserted and lost bits.

Reconstructing Original Message: After obtaining unjammed bits, the receiver still faces several challenges in reconstructing the original message: First, a bit may get lost, if itself and all its copies are jammed by the jammer or mistaken as a false positive. Second, the sender transmits each bit for multiple times, and thus the receiver may receive duplicate bits. In addition, false negatives insert a small number of jammer's bits into the input of the decoder. Therefore, to deal with transmission errors such as inserted, lost, and duplicate bits, the receiver utilizes a *BitTrickle decoder*, which corresponds to the *BitTrickle encoder* used by the sender.

C. Technical Challenges

Detecting (Un)Jammed Bits: Traditional jamming detection aims to find out if wireless communication is jammed (e.g., [16], [19]). However, jamming detection in BitTrickle needs to distinguish jammed bits from unjammed bits. One may suggest the use of received signal strength (RSS) of each bit to distinguish jammed and unjammed bits. Unfortunately, this method will fail when the distribution of RSS values are inherently time-varying due to reasons like the movement of communicators or the use of power control techniques. In Section III, we propose a novel jamming detector that uses modulation properties to distinguish jammed and unjammed bits. The proposed detector does not rely on RSS values, and thus can be used in general wireless applications that have either dynamic or static RSS.

BitTrickle Decoder: Transmission errors such as lost or duplicate bits may happen when there exist jamming attacks or a retransmission mechanism is employed. A small number of lost or duplicate bits can make many bits mis-aligned, which greatly reduce the efficiency of ECC. To deal with lost and duplicate bits, we develop BitTrickle decoder, which can find the original position for each received bit in the encoded message, and enable the use of ECC with high efficiency.

III. Detection of (Un)Jammed Bits

A. Preliminaries on Modulation

I/Q modulation has been widely used in modern wireless systems, including WCDMA, WiMax, ZigBee, WiFi, and DVB (Digital Video Broadcasting). I/Q modulation encodes data bits into physical layer symbols, which are the transmission units in the physical layer. In the following, we use Quadrature Phase-Shift Keying (QPSK) modulation, a typical I/Q modulation, to illustrate how I/Q modulation works.

QPSK – An Example I/Q Modulation: QPSK encodes two bits into one symbol at a time. In Figure 3, bits 00, 01, 10, and 11 are represented by points whose coordinates are $(0, 1)$, $(-1, 0)$, $(0, -1)$, and $(1, 0)$ in an I/Q plane, respectively. The I/Q plane is called a *constellation diagram*. A symbol is the coordinate of a point in the constellation diagram. For a bit sequence 0010, the modulation output are two symbols: $(0, 1)$ and $(0, -1)$. A received symbol is not exactly the same as the original symbol sent by the sender, since wireless channels usually introduce noise to signals that pass through them [8]. To demodulate, the receiver finds the point that is closest to the received symbol in the constellation diagram. For example, in Figure 3, the point closest to the received symbol is (0, 1). Thus, the demodulation output is 00.

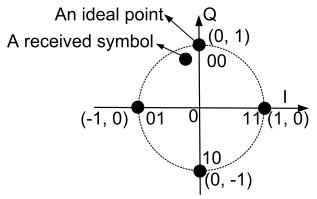

Fig. 3. QPSK modulation/demodulation

B. Observation

Intuitively, jamming signals can introduce a large distortion to signals transmitted by the sender, since the goal of the jammer is to corrupt the signals. If a received symbol is jammed, it may greatly deviate from its ideal point in the constellation diagram and can hardly be recovered. To get more insights in this process, we perform experiments to examine the impacts of jamming on symbol locations.

We collect the received symbols using USRPs [13], which are radio frequency (RF) front ends equipped with analog to digital (AD) and digital to analog (DA) converters. In our experiments, three USRPs are used as the sender, the receiver, and the jammer, respectively, each of which is connected to a computer. Automatic gain control (AGC) is employed by USRPs. We set the bit rate as 1Mbps, carrier frequency as 5GHz, and modulator as QPSK.

We consider a normal scenario and a jamming scenario. In the first one, only the sender transmits randomly generated packets to the receiver, while in the second one, both the sender and the jammer transmit random packets to the receiver concurrently. The receiver record the coordinates of the received symbols in the constellation diagram.

In the normal scenario, as shown in Figure 4, the received symbols form four clusters, each of which centers around an ideal point of QPSK. However, in the jamming scenario, as shown in Figure 5, the received symbols randomly spread over the constellation diagram. Thus, it is hard to identify the ideal

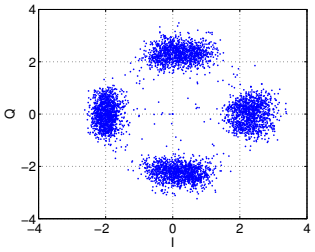

Fig. 4. Normal scenario

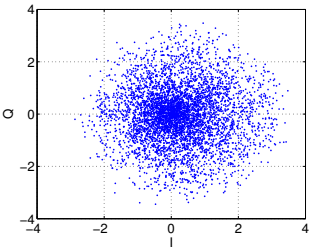

Fig. 5. Jamming scenario

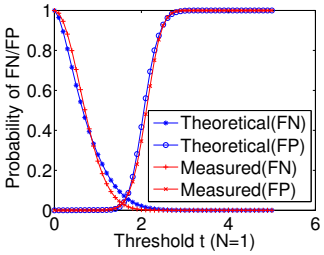

Fig. 6. Theoretical and measured probabilities of false positive/negative when $N = 1$.

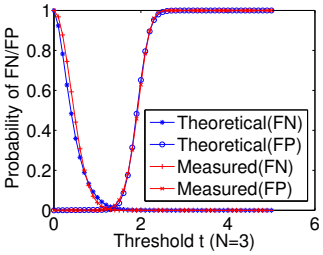

Fig. 7. Theoretical and measured probabilities of false positive/negative when $N = 3$

points for the received symbols, and demodulation errors may happen frequently.

C. Detection Method

Let d_{unjam} (or d_{jam}) be the distance between an unjammed (or a jammed) symbol and the origin in the constellation diagram. As shown in the above experiment, unjammed symbols are close to their ideal constellation points, and thus d_{unjam} approximately equals to the distance between an ideal point and the origin. In contrast, jammed symbols deviate from their ideal points. Due to AGC, such deviation is actually a convergence from ideal points toward the origin rather than an expansion out of the constellation diagram range. Hence, unlike unjammed symbols, jammed symbols are randomly distributed within the constellation diagram, and the expected value of d_{jam} is smaller than that of d_{unjam}. For example, in Figures 4 and 5, the average distance between a received symbol and the origin is 2.2524 and 1.2628, respectively.

We propose to use the distance d between a received symbol and the origin of the constellation diagram as a metric to detect the existence of jammed symbols. For each received symbol, we compute the corresponding distance d, and compare d with a threshold t. If $d > t$, the received symbol is marked as unjammed. Otherwise, it is jammed and we discard it.

Note that different metrics can be explored to accommodate different variants of I/Q modulation. For example, rectangular based I/Q modulation (e.g., 64 QAM) may use the distance between a received symbol and the closest constellation point as the detection metric. We choose the metric d_{unjam} (or d_{jam}) due to its simplicity. This metric serves as an example to illustrate how our observation can be utilized for detecting jammed and unjammed symbols.

The detection accuracy can be enhanced by using the temporal correlation of adjacent symbols. Let s_i and d_i denote the i-th received symbol and its distance from the origin, respectively. We determine whether s_i is jammed or not by examining it along with its neighbor symbols $s_{i-N}, ..., s_{i-1}s_is_{i+1}, ..., s_{i+N}$, where N is system parameter. Symbol s_i is marked as unjammed, if all symbols in this sequence have distances larger than the threshold. As we will show below, this method can enhance the detection accuracy.

D. False Positives and False Negatives

False positives (FP) and false negatives (FN) are two types of errors that may happen in the detection. In a false positive, d_{unjam} of at least one symbol in the temporal sequence is less than or equal to t, and thus an unjammed symbol is incorrectly classified as a jammed symbol. In a false negative, d_{jam} of all symbols in the temporal sequences are larger than t, and thus a jammed symbol is incorrectly classified as an unjammed symbol. Theorems 1 and 2 give both probabilities of false negative and false positive. Proofs can be found in our technical report [11].

Theorem 1: (Probability of false positive) The probability P_{fp} that an unjammed symbol is classified as a jammed symbol is $1-(M_1(\frac{v}{\sigma_N},\frac{t}{\sigma_N}))^{2N+1}$, where M_1 is the Marcum Q-function, v is the distance between an ideal point and the origin of the constellation diagram, t is the threshold, $2N+1$ is the length of the temporal sequence, and ${\sigma_N}^2$ is the variance of the jamming signal.

Theorem 2: (Probability of false negative) Given that each ideal point in the constellation is jammed with equal probability, the probability P_{fn} that a jammed symbol is classified as an unjammed symbol is $(e^{\frac{-t^2}{2\sigma^2}})^{2N+1}$, where t is the threshold, $2N+1$ is the length of the temporal sequence, and σ^2 is the variance of the I/Q coordinate of a received symbol.

Experimental Validation: To verify the theoretical results, we run the temporal check enhanced method to detect unjammed symbols from symbols collected for normal and jamming scenarios in our earlier experiment. The measured false positive probability P_{fp} and false negative probability P_{fn} are computed by $P_{fp} = 1 - \frac{\text{\# detected symbols}}{\text{\# total symbols}}$ and $P_{fn} = \frac{\text{\# detected symbols}}{\text{\# total symbols}}$, respectively. Meanwhile, we compute P_{fp} and P_{fn} using Theorems 1 and 2. The computation results are shown in Figures 6 and 7. Note that statistic parameters v, σ_N, and σ are determined based on our earlier experiment[1]. Both theoretical and real measured results are in close consistency. A large N can result in both small P_{fn} and P_{fp}. When $N = 1$, both real measured P_{fn} and P_{fp} can be as low as 0.0444 by using a threshold t that equals to 1.6. If we increase N to 3, we can achieve even lower error rate.

E. Determining the Threshold

The threshold t can be determined based on the system requirement for P_{fn} and P_{fp}. For example, if the false negative probability P_{fn} is required to be less than α, we have $(e^{\frac{-t^2}{2\sigma^2}})^{2N+1}<\alpha$. By treating t as an unknown and solve the inequality, we can get $t>\sqrt{2\sigma^2 \ln \alpha^{2N+1}}$. As the threshold

[1] $v = 2.3949$, $\sigma_N = 0.3838$, and $\sigma = 1.0344$

t increases, P_{fp} increases but P_{fn} decreases. If the goal is to minimize both P_{fp} and P_{fn}, as shown in Figures 6 and 7, the minimization result and the corresponding t form the intersection point of the P_{fp} and P_{fn} curves.

IV. BitTrickle Encoding/Decoding

The original message is first encoded with a traditional ECC (e.g., Reed-Solomon codes) before being processed by BitTrickle. ECC corrects substitution errors (i.e., bit "'1" is replaced by "0" and vice-versa). The BitTrickle encoding scheme further encodes the ECC-coded message to allow a receiver to decode the correct positions of received bits and recover from synchronization errors.

A. Basic Idea

For the sake of presentation, we call the input to BitTrickle encoding (i.e., the ECC-coded message) as a BTmessage.

BitTrickle Encoding: The sender and the receiver agree on a sequence that is formed by n integers, where n is the length of the BTmessage. We call such an integer sequence a *positioning code* and each integer in the sequence a *label*. As shown in Figure 8, the BTmessage is 10110 and the positioning code is 03572. For $1 \leq i \leq 5$, the sender labels the i-th bit of the message using the i-th label in the positioning code (e.g., the second bit is 0 and its label is 3). In the labeling, the sender uses one symbol to represent both a bit and its label. (Details of labeling will be presented in Section IV-B.) Note that a symbol is the transmission unit of physical layer. Once a receiver receives a symbol, the receiver knows both the bit and its label. The encoding results are shown in Figure 8.

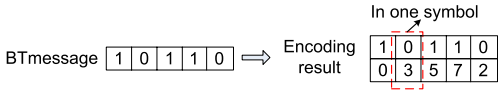

Fig. 8. BitTrickle encoding

Transmission Errors: The sender takes random backoffs or performs channel sensing to avoid colliding with the jammer. Without loss of generality, we assume the sender adopts the backoff based method. Figure 9 shows an example. The sender transmits the first symbol for 3 times, takes a random backoff, and transmits this symbol again for 3 times. The sender repeats until the last symbol is transmitted. Due to jamming and retransmissions, a symbol may get lost or duplicated. In Figure 9, all copies of the 2nd symbol are lost and the 4th symbol is duplicated. Also, there may exist a small amount of inserted symbols caused by false negatives of the authenticator.

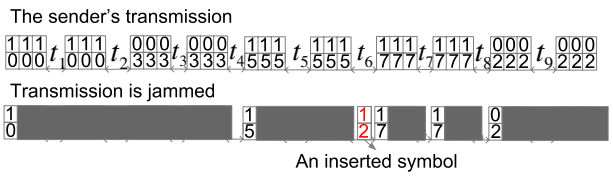

Fig. 9. Transmission Errors

BitTrickle Decoding: The receiver demodulates each received symbol to extract the bit and corresponding label carried by this symbol. Figure 10 shows an example following Figure 9. The extracted bits and labels are 111110 and 052772, respectively. The receiver then takes two steps to correct synchronization errors.

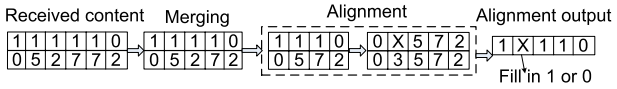

Fig. 10. BitTrickle decoding

The first step is merging, in which bits are merged into a single bit if they are identical and have the same label. As shown in Figure 10, the 4th and the 5th received bit are identical (i.e., both of them are 1), and have the same label 7. Thus, they are merged together. The merging result is 11110 and the corresponding labels are 05272. An incorrect merging may happen if multiple bits in the BTmessage are identical and use the same label. In Section IV-C, we give the analytical upper bound of the error probability, and show that the upper bound decreases quickly as configurable parameters such as the number of retransmissions increases. The second step is alignment, which consists of two substeps:

(1) Dealing with False Negatives: Although most fake symbols can be identified by the physical layer authenticator, a small amount of them may survive due to false negatives of the authenticator. Those symbols are actually incoherent pieces of the fake symbol stream and the correlation between their labels and the positioning code is weak. Therefore, to filter out inserted bits, we perform alignment on the most correlated part between the positioning code and the merged received labels. We find the largest common subsequence (LCS) between the positioning code and received labels, and align the LCS with the positioning code. For example, in Figure 10, the received labels are $05\underline{2}72$, where the underlined 2 is the inserted label from the jammer. The LCS between $05\underline{2}72$ and the positioning code 03572 is 0572. Thus, the inserted label is filtered.

LCS is not necessarily unique. Finding all LCSs requires exponential time complexity in the worst case, whereas finding one LCS is solvable in polynomial time by dynamic programming [6]. Therefore, we utilize existing dynamic programming method to only find one LCS. It is possible that there exist multiple LCSs and the LCS returned by dynamic programming contains inserted labels. However, as shown in the appendix of [11], such probability decreases quickly with the increase of the percentage of the sender's labels. Since false negatives are rare events, the sender's labels comprise the great majority of total received labels, and thus there is a high chance that the sender's labels form the LCS of received labels and positioning code. Retransmissions further increase this chance. For example, the sender may transmit a BTmessge for 3 times. For each transmission, the receiver can obtain a LCS. Given a 0.1 probability that a LCS contains inserted bits, the chance that at least one LCS does not contain inserted bits is 0.999.

(2) Generating Alignment Output: In the LCS 0572, the labels 0, 5, 7, and 2 match the 1st, 3rd, 4th, and the last label in the positioning code, respectively. Thus, the receiver knows that the second bit is lost, and corrects synchronization

errors by filling a bit that can be either 1 or 0 in the position shown in Figure 10. The alignment output is further processed by traditional ECC to recover the original message. There may exist multiple alignment outputs. In Section IV-C, we develop a fast alignment approach that not only achieves desired alignment accuracy, but also reduces the overhead by only trying a subset of all combinations.

Diversity Degree of a Positioning Code: To avoid incorrect merging/alignment, we require that consecutive labels in the positioning code to be different. Specifically, the i-th label in the positioning code does not equal to any of its previous d labels (i.e., $i-1$-th, ..., $i-d$-th label) and successive d labels (i.e., $i+1$-th, ..., $i+d$-th label), where $d \geq 1$ is an adjustable parameter, referred to as *diversity degree* of the positioning code. For example, when diversity degree is 2, the 8th label should not be the same as the 7th, 6th, 9th and 10th label.

B. Encoding at Sender

The sender adds special data content (e.g., 11111) to both the beginning and the end of a BTmessage, so that a receiver can recognize the boundary of a BTmessage. We refer to the special data content as a *message delimitation code (MDC)*.

Afterwards, the sender labels the i-th bit of the BTmessage by packing the i-th bit and the i-th label of the positioning code into one physical layer symbol. For example, assume that i-th bit is 1 and its label is 2. The sender appends 10 (i.e, binary form of 2) to the data bit 1, and the result is 1$\underline{10}$, which are modulated into one symbol (e.g., a 8PSK symbol). To improve efficiency, bits in the MDC are not labeled. For an M-ary modulator that encodes $\log_2 M$ bits by one symbol, the maximum value of a label of the positioning code should be $2^{\log_2 M-1}-1 = \frac{M}{2}-1$. For example, an 8PSK symbol uses one bit to carry data information and two bits to carry the label. Hence, a label is less than or equal to 3 (i.e., 11). Packing a data bit and its label in one symbol achieves atomicity: data bits are always associated with their labels. Upon receiving a symbol, the receiver knows both the data bit and its label.

C. Decoding at Receiver

Before decoding, the receiver searches for boundaries of a BTmessage. The boundary of the BTmessage is identified if the receiver can observe an MDC or a certain data pattern that is a part of MDC. For example, assume that the MDC equals to 1111111, the receiver identifies the beginning or end of a BTmessage if the receiver receives 1111111, multiple consecutive 1's (e.g., 1111), or multiple consecutive 1's interleaved with quite a few 0's (e.g., 1110111). The third condition deals with bits inserted by false negatives. Note that most fake MDCs injected by the jammer have already been filtered out by the physical layer authenticator.

To reduces the chances that the entire MDC is jammed, the sender and the receiver can increase the length of the MDC according to the severity of jamming attacks, so that the receiver can observe at least a part of the MDC. Alternatively, they may take a backoff time between transmitting two consecutive symbols of an MDC.

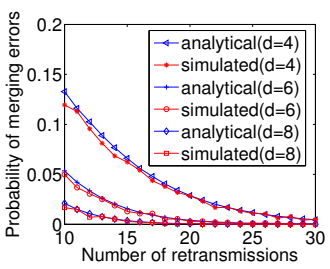

Fig. 11. Merging errors

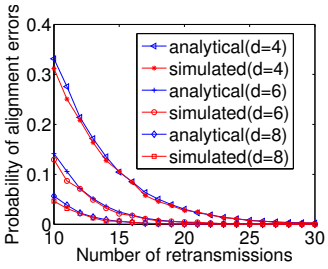

Fig. 12. Alignment errors

The receiver then demodulates the symbols of the received BTmessage, extracts a data bit and a label from each symbol, and takes two steps to correct synchronization errors.

Merging: Bits are identified as duplicated and merged into a single bit if they are consecutive, identical and have the same label. To detect and merge duplicated bits, the receiver points a cursor to the first bit/label of the received BTmessage. Then, the receiver compares the bit/label pointed by the cursor and each of the following $N_r - 1$ bits/labels, where N_r denote the number of retransmissions for a single bit. If inequality occur (e.g, two bits are not equal or have different labels), the receiver merges all equal bits/labels together and points the cursor to the next bit/label. The receiver repeats until all bits/labels of the received BTmessage are scanned.

Merging Errors: Different bits may be incorrectly merged together. Theorem 3 give the upper bound of the probability of merging errors. The proof can be found in [11].

Theorem 3: (Probability of merging errors) The probability p_e that a received BTmessage is merged incorrectly is less than $1-(1-\frac{p^{rd}-p^{r(n-n(1-p^r)+1)}}{2(R-d)})^{n(1-p^r)-1}$, where n is the length of a positioning code, R is the number of possible values for each label, r is the number of retransmissions for each bit in the BTmessage, d is the diversity degree of the positioning code, and p is the probability that a bit transmitted is lost.

We use simulations to validate the analytical upper bound. All simulations are done in MATLAB 7.7.0. We let $R = 32$ and $p = 0.95$, and perform 10,000 trails. In each trial, we randomly generate a message and a positioning code of length 155, and label the message using the positioning code. We retransmit each bit of the message for r times ($10 \leq r \leq 30$), and delete each retransmitted bit with probability p. We then merge the remaining bits, and compare the result with the correct result obtained based on the original generated message. If both results are not equal, a merging error happens and we mark this trial as failed. We compute the simulated probability of merging error and its analytical upper bound using ($\frac{\text{\# failed trials}}{\text{\# total trials}}$) and Theorem 3, respectively.

As shown in Figure 11, the simulated probability is only slightly less than its analytical upper bound, which indicates that the result Theorem 3 is a tight upper bound. A larger diversity degree d can achieve smaller error probability. As the number r of retransmissions increases, both the simulated probability and its upper bound decrease and approach to 0. In particular, when $d = 8$ and $r = 20$, the simulated probability and the analytical upper bound are 0.0005 and 0.0006.

Merging errors will generate additional lost bits during

message recovery. In the following, we develop a method to recover lost bits through alignment and ECC.

Alignment: The goal of alignment is to find the actual position of each received bit in the original BTmessage. Let S denote the positioning code and L the merged labels (e.g., in Figure 10, the merged labels are 0572).

(1) Basic Alignment Method: If the length of the positioning code is small, we can do alignment in a brute force way. Specifically, assume that the length of L is q. The receiver can find all length-q subsequences of S, and compare each of them with L. For each subsequence that equals to L, the receiver generates an alignment output by padding 1's or 0's into the positions of lost bits. For example, assume that padding bits are 1's and the received message after merging is 00. For $L = 17$ and $S = 1317$, the alignment outputs are 0110 and 1100. Each alignment output is further processed by traditional ECC decoding, where replacement errors (i.e., $1 \rightarrow 0$ or $0 \rightarrow 1$) are corrected. Since there may exist multiple alignment outputs, the receiver may obtain multiple decoding results, among which the one that can pass cyclic redundancy check (CRC) or authentication is the recovered message. If the length of S is large, this method is time consuming. We develop a fast alignment approach below to reduce the overhead.

(2) A Fast Alignment Method: To achieve fast alignment, we propose to only find one alignment. We further show that given proper configurations, this single alignment leads to a very small error probability. We use a simple greedy strategy to obtain a single alignment. Specifically, the receiver compares labels of L with those of the positioning code S, trying to find S's leftmost or rightmost subsequence that equals to L. For example, if $L = 17$ and $S = 1177$, the S's leftmost and rightmost subsequence that equals to L is underlined in $\underline{1}1\underline{7}7$ and $1\underline{1}7\underline{7}$, respectively. The positions of the leftmost/rightmost subsequence is 13/24, and thus the corresponding decision is that the first and the third bits of the BTmessage are received (or the second and the last bits are received).

Alignment Errors: For basic alignment, the probability that alignment errors happen is 0. For fast alignment, alignment errors may happen if the positions of the leftmost/rightmost subsequence are not formed by the correct positions of the received bits. Theorem 4 gives the upper bound of the probability of alignment errors. The proof can be found in [11].

Theorem 4: (Probability of alignment errors) The probability p_e that the receiver fails to generate correct alignments is $1 - \sum_{k=q}^{n} \frac{\binom{k}{q}}{\sum_{w=q}^{n}\binom{w}{q}}(1 - \frac{p^{rd} - p^{r(n-q+1)}}{R-d})^{k-q}$, where n is the length of a positioning code, R is the number of possible values for each label, r is the number of retransmissions for each bit in the BTmessage, d is the diversity degree of the positioning code, and p is the probability that a bit is lost.

We also use simulation to validate the analytical upper bound of alignment errors. The parameters are the same as those used in the simulation for merging errors (i.e., $R = 32$, $p = 0.95$, and 10,000 trials). In each trial, we randomly generate a positioning code of length 155, retransmit each label for r times ($10 \leq r \leq 30$), and delete each retransmitted label with probability p. The remaining labels are merged together. Then we find the positions of received bits (labels) using the fast alignment alignment approach, and compare the result with the true positions. If they are not equal, an alignment error happens and we mark this trial as failed. We compute the simulated probability of alignment error and its analytical upper bound using ($\frac{\text{\# failed trials}}{\text{\# total trials}}$) and Theorem 4.

Figure 12 shows that the error probability decreases as diversity degree d and the number of retransmissions increase. From Figure 12, we can observe that theorem 4 gives a tight upper bound of the error probability. In particular, when $d = 8$ and $r = 20$, both the simulated probability and the upper bound are about 0.0006.

V. Implementation and Evaluation

We develop a prototype system for BitTrickle to facilitate the experimental evaluation of BitTrickle performance under reactive jamming. The prototype system consists of a sender and a receiver, both implemented as a USRP connected to a commodity PC that runs the sender (receiver) program. The USRPs uses XCVR2450 daughter boards operating in the 2.4GHZ range as RF front ends. The software implementing BitTrickle is based on GNURadio [1].

Reactive Jammer: We setup a high power and sensitive reactive jammer to test the performance of BitTrickle. The jammer is implemented on USRPs using GNURadio [1]. We employ energy detection to achieve a lower channel sensing time (i.e., a signal is detected if received signal strength exceeds a configurable threshold). In our design of the jammer, we equip the jammer with two RFX2400 daughter boards that are used as a transmitter and a receiver, respectively. For both the transmitter and receiver component, we set the parameter "samples per symbol" the minimum value supported by GNURadio to reduce the processing delay (i.e., 2 and 4 for transmitter and receiver, respectively). Also, to maximize the impact of the jammer on the BitTrickle receiver, we let the jammer transmits with maximum gain and place the jammer very close to the receiver (i.e., within 0.1 meter range of the receiver). Parameters of the jammer is shown in Table I.

TABLE I
Technical details of the reactive jammer

Parameter	Value
Frequency range	2.3 – 2.9 GHz
Channel sensing time	0.6 ms
Transmit power	50 mW
Interpolation/Decimation rate	64/32
Maximum receiving RF bandwidth	16 MHz

Compared Schemes: We compare the following schemes: (1) **BitTrickle**: The prototype implementation of BitTrickle sender and receiver. This approach uses Reed-Solomon (RS) error correction codes, and differential 8PSK modulator/demodulater. The prototype system supports two RS coding rates, which are RS(155, 55) and RS(60, 36). (2) **GNURadio Benchmark**: The communication tool provided by GNURadio for data transmission and file transfer between two USRPs. The source codes are located at gnuradio/gnuradio-examples/python/digital. (3) **802.11 DSSS**: IEEE 802.11 protocol running at direct-sequence spread spectrum (DSSS) mode on

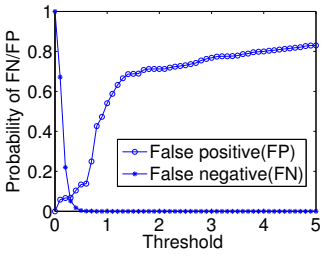

Fig. 13. Jamming detector

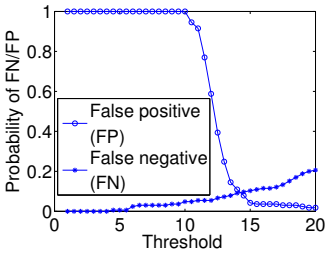

Fig. 14. Authenticator

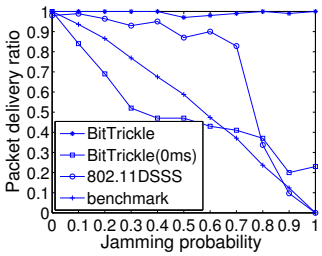

Fig. 15. Packet delivery ratio

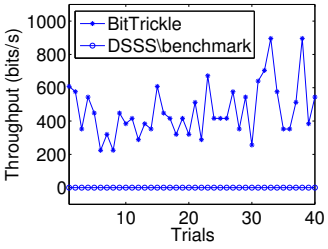

Fig. 16. Throughput

802.11 wireless cards. This approach uses a 11-bits barker code for spreading, carrier sense multiple access with collision avoidance mechanism (CSMA/CA) to resolve collisions on shared channels, and forward error correction (FEC) to enable the reconstruction of the original data.

Evaluation Metrics: A jammer aims to prevent the communication between legitimate users. Therefore, how well the sender and the receiver can communicate under jamming attacks is a primary concern to assess anti-jamming systems. We use the following metrics to evaluate the performance: (1) **Packet delivery ratio:** The ratio of the number of correctly received packets to the total number of packets transmitted by the sender. We consider a packet to be received correctly if the packet passes CRC check. (2) **Throughput:** This is the number of successfully delivered bits normalized by time unit. We use bits per second to measure the throughput.

A. Component Evaluation

Jamming Detector: The function of jamming detector is to remove jammed symbols. We implement the temporal based detection method discussed in Section III-C to detect jammed symbols. To evaluate false negative rate and false positive rate, we also perform off-line analysis in MATLAB. Figure 13 shows the result for $N = 5$, where N denotes the temporal sequence length. We can see that a threshold of 0.3 balances the false negative and false positive.

Physical Layer Authenticator: We implement a simple device authenticator based on [4], which uses modulation error metrics (i.e., frequency error, phase error, magnitude error, EVM, I/Q offset, SYNC correlation) to identify wireless devices. To simplify the implementation of the authenticator, we only choose EVM (error vector magnitude) as the metric to identify the sender. In the training stage (the jammer is turned off), we let the sender transmit and record the EVMs of received symbols. Those EVMs are used as the fingerprints of the sender's signal. The receiver computes the Euclidean distance between received k symbols and each of the fingerprints. The minimum distance is then compared with a threshold to decide if the received symbols are from the sender. To test false negative and false positive rate, we again perform off-line analysis in MATLAB. Figure 14 shows the result. For a threshold that equals to 14.5, the authenticator can achieve a 0.0970 false negative rate and a 0.0788 false positive rate. Noted that we use a very simple physical layer authenticator in our prototype system. Certainly other advanced authenticators can be used to get an even lower false rate and higher security.

B. Performance of BitTrickle

We set the bit rate of the sender, the jammer, and the receiver to be 1Mbps. The sender transmits 100 data packets, each with 1500 bytes. Positioning codes are randomly generated, and the diversity degree is set to 2 throughout the evaluation. Since the size of a data packet (1,500 bytes) is too long to be directly used with the positioning code and ECC, we divide it into multiple blocks and append a CRC checksum to each block. We use block size 36 or 55 bits, then RS(60,36) or RS(155, 55) for ECC, and finally a positioning code of 60 or 155 bits.

Packet Delivery Ratio: We consider different jamming intensities. We use a probabilistic reactive jammer that jams at probability p for $0 \leq p \leq 1$ once detects a sender's signal. The jamming duration is set to be 10 times of the transmission time of a single packet. We compute packet delivery ratio as $\frac{\text{\# correct packets (blocks)}}{\text{\# total transmitted packets (blocks)}}$. Figure 15 shows the result.

(1) 802.11 DSSS and GNURadio Benchmark: Packet delivery ratio decreases as jamming probability increases. Due to the lack of ECC and retransmission mechanism, the packet delivery ratio of GNURadio benchmark decreases at a rate linearly proportional to the jamming probability. Although 802.11 DSSS achieves a higher packet delivery ratio than GNURadio benchmark, when jamming probability exceeds 0.7, the performance of 802.11 DSSS degrades dramatically. For both 802.11 and benchmark, when the jamming probability equal to 1, the packet delivery ratio drops to 0.

(2) BitTrickle: We use a random backoff ranging between 150–200 ms and set the number of bit retransmissions to be 15. Figure 15 shows that BitTrickle achieves a stable packet delivery ratio that is around 1 no matter how the jamming probability varies. We then reduce the backoff time to 0 ms and increase the bit retransmissions to 60. Figure 15 shows that the packet delivery ratio of BitTrickle decreases as jamming probability increases. This is because the reduced backoff time increases the chance that the sender's signal collides with the jammer's signal. The modulator used by the BitTrickle prototype has a higher bit error rate (i.e., BER) than that used by GNURadio benchmark (i.e., GFSK). Therefore, the packet delivery ratio of BitTrickle is less than that of benchmark when jamming probability is small (e.g., ≤ 0.7). However, when the probability is 1, unlike benchmark and 802.11, BitTrickle with zero backoff still achieves a non-zero delivery ratio.

Throughput: We consider a common jamming scenario, where the reactive jammer jams the channel as long as it hears the target signal (i.e., $p = 1$). To be conservative, we set the backoff time of BitTrickle to be 0 ms. We performs 40 trials. In

each trial, the number of bit retransmissions is set to 60 and the sender transmits 100 data packets to the receiver. We compute throughput as $\frac{\text{\# correct packets (blocks)} \times \text{packet (block) length}}{\text{transmission time}}$. Figure 16 plots the computed throughput for each trial. The GNURadio benchmark and 802.11 DSSS fail to send any packet, whereas BitTrickle still achieves a throughput that ranges between 200–900 bits/s, allowing communication to continue.

We also test the BitTrickle throughput under different ECC coding rate (i.e., RS(155,55) and RS(60,36)). Figure 17 plots the throughput as a function of signal-to-jamming ratio (SJR) (i.e., the ratio of the reaction time to jamming duration). As

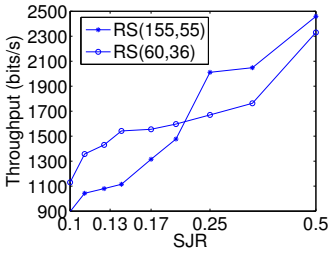

Fig. 17. Throughput for different coding rate.

shown in Figure 17, RS(60,36) leads to a higher throughput than RS(155,55) when SJR is less than 0.25. That's because RS(60,36) requires a shorter positioning code than RS(155,55), which reduces the chance of synchronization errors. As SJR increases, the receiver gets more information from the sender, and thus the probability of synchronization errors decreases. The error correction capability of RS(155,55) is stronger than that of RS(60,36). For small SJRs, RS(155,55) does not suffer from severe synchronization errors, and thus it can correct more substitution errors and achieve a better throughput.

VI. Related Work

FHSS and DSSS (e.g., [8], [12], [15]) have been widely used for jamming defense. However, they cannot defend against broadband or high power jammers. A recent paper considers threats from broadband jammers, and proposes to use timing-based covert channels to address broadband jammers [18]. The idea is to map the inter-arrival times of a sender's corrupted packets into information bits [10]. This method fails if the jammer launches pollution attacks or transmits with high power to overwhelm transmitted packets. Our work considers both broadband and high power jammers, as well as pollution attacks. Our work is also related to reactive jamming detection. Strasser et al proposed to detect jamming attacks by using the correlation between corrupted bits and the corresponding RSS [16]. However, [16] aims to identify the cause of bit errors for individual packets, whereas the jamming detector in this paper aims to distinguish unjammed bits from jammed bits.

VII. Conclusion

We developed BitTrickle to enable wireless communication when a broadband and high power reactive jammer is present. BitTrickle delivers information by harnessing the reaction time of a reactive jammer. It does not assume a reactive jammer with limited spectrum coverage and transmit power, and thus can be used in scenarios where traditional approaches fail. We implemented a prototype of BitTrickle based on GNU Radio. Our results showed that BitTrickle achieved a reasonable throughput when 802.11 DSSS and GNURadio benchmark were completely disabled by the jammer.

Acknowledgements

This work is supported by the National Science Foundation (NSF) under grant CNS-1016260. The authors would also like to thank the anonymous reviewers for their helpful comments.

References

[1] GNU Radio - The GNU Software Radio. http://www.gnu.org/software/gnuradio/.

[2] Reactive jamming technologies. http://www.ece.gatech.edu/academic/courses/ece4007/08fall/ece4007l02/lm5/jammer.doc.

[3] *Jane's Military Communications, Edition 22, 2001–2002*. Jane's Information Group INC, Virginia, USA, 2002.

[4] V. Brik, S. Banerjee, M. Gruteser, and S. Oh. Wireless device identification with radiometric signatures. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 116–127, New York, NY, USA, 2008. ACM.

[5] D. Cabric, A. Tkachenko, and R. W. Brodersen. Experimental study of spectrum sensing based on energy detection and network cooperation. In *TAPAS '06: Proceedings of the First International Workshop on Technology and Policy for Accessing Spectrum*, page 12, New York, NY, USA, 2006. ACM.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd*. MIT Press, 2001.

[7] B. Danev, H. Luecken, S. Capkun, and K. E. Defrawy. Attacks on physical-layer identification. In *Procceedings of the 3nd ACM Conference on Wireless Networking Security (WiSec '10)*, pages 89–98, March 2010.

[8] A. Goldsmith. *Wireless Communications*. Cambridge University Press, August 2005.

[9] H. Kim and K. G. Shin. In-band spectrum sensing in cognitive radio networks: energy detection or feature detection? In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 14–25, 2008.

[10] L. Lazos, S. Liu, and M. Krunz. Mitigating control-channel jamming attacks in multi-channel ad hoc networks. In *Proceedings of 2nd ACM Conference on Wireless Networking Security (WiSec '09)*, March 2009.

[11] Y. Liu and P. Ning. Bittrickle: Defending against broadband and high-power reactive jamming attacks. Technical Report TR-2011-17, NC State University, Computer Science Department, July 2011.

[12] Y. Liu, P. Ning, H. Dai, and A. Liu. Randomized differential dsss: Jamming-resistant wireless broadcast communication. In *Proceedings of the 2010 IEEE INFOCOM*, 2010.

[13] Ettus Research LLC. The USRP product family products and daughter boards. http://www.ettus.com/products. Accessed in April 2011.

[14] N. Patwari and S. K. Kasera. Robust location distinction using temporal link signatures. In *MobiCom '07: Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 111–122, New York, NY, USA, 2007. ACM.

[15] Robert A. Scholtz. *Spread Spectrum Communications Handbook*. McGraw-Hill, 2001.

[16] M. Strasser, B. Danve, and S. Capkun. Detection of reactive jamming in sensor networks. *ACM Transaction on Sensor Networks*, 7, Aug. 2010.

[17] M. Strasser, C. Pöper, S. Čapkun, and M. Čagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, 2008.

[18] W. Xu, W. Trappe, and Y. Zhang. Anti-jamming timing channels for wireless networks. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 203–213, New York, NY, USA, 2008. ACM.

[19] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '05)*, 2005.

[20] R. Zhang, Y. Zhang, and X. Huang. JR-SND: Jamming-resilient secure neighbor discovery in mobile ad hoc networks. In *Procceedings of the 32nd International Conference on Distributed Computing Systems (ICDCS'11)*, June 2011.