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—**A reactive jammer jams the wireless channel only when the target devices are transmitting. Reactive jamming is not only cost effective, but also are 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 as countermeasures 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 two novel techniques in BitTrickle, including an approach based on modulation errors to detecting and extracting unjammed bits from partially corrupted packets, and an encoding/decoding method to recover the original message from a collection of message fragments whose positions in the original message are unknown. We develop a prototype of BitTrickle using the USRP platform running GNURadio. Our experimental evaluation shows that when under powerful reactive jamming, BitTrickle still maintains communication, whereas other schemes such as 802.11 DSSS fail completely.**

I. Introduction

Reactive jamming is one of the most effective jamming attacks [3]. 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 are hard to track and remove due to its intermittent jamming behaviors [3], [38]. Reactive jamming has been widely used in military applications to cut off the wireless communication of the enemy army or disable radio-controlled devices [3].

Current defense against jamming attacks mainly relies on spread spectrum techniques, which are categorized as Frequency Hopping Spread Spectrum (FHSS) (e.g., [14], [28], [31], [32], [34]), Direct Sequence Spread Spectrum (DSSS) (e.g., [14], [20], [24], [28], [34]), Time Hopping Spread Spectrum (THSS) (e.g., [27], [37]), and Chirp Spread Spectrum (CSS) (e.g., [6], [15], [29]). Among these techniques, FHSS and DSSS are dominantly used for anti-jamming purposes, whereas THSS and CSS require specific hardware (e.g., a pulse/chirp generator) [27], [35], and thus are less frequently used.

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 will fail to maintain communication.

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

In this paper, we develop BitTrickle, a novel wireless 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 employ BitTrickle as a countermeasure against 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" [41].

Channel sensing is an indispensable function 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 [16], requires more than 1 millisecond 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 (field programmable gate array) for fast speed [8]. Therefore, before the jammer detects the sender's transmission and starts jamming, the sender has already transmitted one or several bits. As shown in Figure 1, though the packet transmitted by the sender is corrupted and cannot be fully recovered, the first few bits of this packet are unjammed due to channel sensing delay at the jammer.

BitTrickle exploits the unjammed bits in corrupted packets to establish jamming-resilient communications. In BitTrickle, 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 jamming, in which the jammer sleeps after jamming for a random time

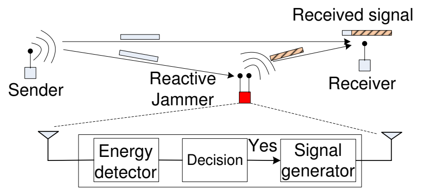

Fig. 1. Reactive jamming: The first few bits are not jammed due to jammer's channel sensing delay.

and resumes afterwards.

Two technical challenges are addressed during the development of BitTrickle. First, the receiver needs to extract unjammed bits from partially corrupted 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 deliver the similar performance provided by traditional error correction codes (ECC). Accordingly, we develop a novel encoding and decoding technique to locate the original positions for unjammed bits and enable error recovery with high efficiency.

Note that the bandwidth of BitTrickle partially depends on the reactive jamming pattern. Longer jamming duration will lower BitTrickle's bandwidth. (This is similar to FHSS and DSSS, which have lower bit rates to tolerate stronger jamming attacks that cover more channels with higher power.) However, at the same time, the reactive jammer will risk 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. In scenarios where 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 based anti-jamming systems typically support a bit rate from dozens bps to several kilo bps [4]. For example, PRC 3100H radio systems, which are FHSS based transceivers used by US Army to provide encrypted voice and data communication in battlefields, can achieve a bit rate up to 2400bps with error correction [4]. Our prototype implementation of BitTrickle on the Universal Software Radio Peripheral (USRP) platform [21] achieves a data rate between 200-2,500 bps, which can support the above tactical communication under attacks.

Our contribution in this paper is three-fold. First, to defend against broadband, high-power reactive jamming attacks, we develop BitTrickle by exploiting jammer's sensing delay, which enables wireless communication even when previous anti-jamming techniques fail. Second, we develop two novel techniques in BitTrickle, including a modulation error based method to extract unjammed bits from partially corrupted packets, and an encoding and decoding method to recover the original message from message fragments whose positions in the original message are unknown. Finally, we implement a BitTrickle prototype using the USRP platform [21] running GNURadio [1], and compare it with 802.11 DSSS and a benchmark program in GNURadio. Our evaluation results show that BitTrickle achieves a reasonable throughput when the other approaches are completely disabled by a reactive jammer.

The rest of the paper is organized as follows. Section II clarifies the assumptions and threat model. Section III gives an overview of BitTrickle. Sections IV and V present our methods for unjammed bits extraction and final message reconstruction. Section VI describes the implementation and experimental evaluation. Section VII discusses related work, and finally Section VIII concludes this paper.

II. Assumptions and Threat Model

Our system consists of a sender and a receiver, who aim to communicate in the presence of jamming attacks. We assume that both the sender and the receiver are general wireless devices that can transmit and receive wireless signals. We assume an *intelligent reactive* jammer who emits noisy signals onto wireless channels to interfere with the transmission from the sender to the receiver. The goal of the jammer is to block the communication from the sender to the receiver efficiently. Specifically, the jammer remains quiet when the sender is not transmitting, but starts sending noise signal once it senses any transmission from the sender. We assume that the jammer has a high transmit power and can jam all channels. One constraint is that the jammer cannot break the employed authentication mechanism.

III. Overview of BitTrickle

As discussed earlier, BitTrickle exploits the sensing delay of reactive jamming to enable message transmission. The key to BitTrickle is thus how to encode and decode meaningful messages assuming a major part of each transmission is corrupted.

Though BitTrickle is targeted at broadband high-power reactive jammers, it can also defend against random jamming attacks. In both reactive and random jamming scenarios, a common feature is that multiple transmitted bits are lost. The BitTrickle techniques can be used to extract the unjammed bits and reconstruct the original messages in both cases.

In the following, 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 uses a *BitTrickle encoder* to encode a message, which enables the receiver to identify the boundary of a received encoded message and recover the message in the presence of partial message corruption. To reduce the time for the jammer to observe and react to messages, the sender should use a short preamble so that after the reactive jammer detect the preamble there is enough time for at least one bit data be to transmitted before the jammer start jamming the rest of the message. The sender also transmits each bit of the encoded message for multiple times to increase the chance that the receiver can receive this bit.

The sender needs to find times when the jammer is not jamming to transmit messages. The sender may take a random

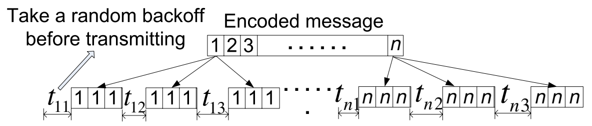
Take a random backoff before transmitting
Encoded message
1 2 3 ...... n
t_{11} 1 1 1 t_{12} 1 1 1 t_{13} 1 1 1 ...... t_{n1} n n n t_{n2} n n n t_{n3} n n n

Fig. 2. Transmission at the sender

backoff before each transmission, as shown in Figure 2. This makes it hard for the reactive jammer to predict when the sender will start the next transmission. The 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 (i.e., the sender can hear the jammer's signals), 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 to the air without waiting for the backoff time to expire.

Figure 2 shows the most conservative situation, where the sender has no information on the reception at the receiver. Thus, the sender attempts to transmit one bit a time. The performance can be improved by learning how many bits can be received in one transmission 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. As a result, the sender can transmit multiple bits a time. In this paper, we focus 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 that survive the reactive jamming and reconstruct the original message from these unjammed bits, which are possibly collected from multiple transmissions.

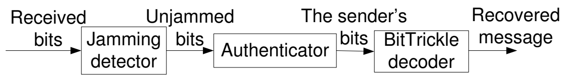

Fig. 3. Reception at the Receiver

Extracting Unjammed Bits: Figure 3 shows the high-level view of the receiver's operations. The receiver first processes each received bit with the *jamming detector*, which checks whether this bit is jammed or not, and discard all the jammed bits. The output of the jamming detector is thus a collection of unjammed bits, which are possibly extracted from multiple transmissions.

Dealing with Pollution Attacks: An intelligent jammer who knows our scheme may attempt to pollute the unjammed bits to defeat our scheme. Specifically, the jammer may transmit 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., [7]) and radio frequency (RF) fingerprints (e.g., [23], [43]). As cryptographic authentication faces cryptanalysis based attacks, physical layer authentication may also face similar threats as revealed by [13]. 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 slight amount of inserted bits and lost bits. In Section V, 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 a false positive happens. Second, the sender transmits each bit for multiple times, and thus the receiver may receive duplicate bits. In addition, false negatives insert a small amount 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

Two technical problems need to be solved to develop BitTrickle.

Detecting (Un)Jammed Bits: The requirement of jamming detection in BitTrickle is drastically different from traditional jamming detection. Instead of finding out whether or not wireless communication is jammed (e.g., [30], [41]), jamming detection in BitTrickle has to distinguish jammed bits from unjammed ones. This requirement makes the previous jamming detectors insufficient.

One may suggest the use of received signal strength (RSS) of each bit to distinguish jammed and unjammed bits, i.e., identify a jammed bit when its RSS is high. However, this method will fail since the distribution of RSS values may be time-varying. For example, RSS values increase as a sender gets near to a receiver and decrease as the sender moves away. Similarly, for wireless devices that utilize power control technologies (e.g., [12], [17], [25]), jamming is not the only reason that causes a high RSS values, since the sender dynamically adjusts its power for efficiency. In the next section, we propose a novel technique that uses modulation properties to distinguish between jammed and unjammed bits.

BitTrickle Decoder: Transmission errors like lost or duplicate bits may happen when there exist jamming attacks or a retransmission mechanism is employed. At first glance, an ECC might be directly used by BitTrickle to deal with transmission errors. However, a closer look reveals that this message reconstruction is quite different from traditional error recovery, where both correct and error bits are in their correct positions. Indeed, in a sequence of bits received by the BitTrickle Decoder, a small number of lost or duplicate bits can make many bits mis-aligned, which greatly reduce the efficiency of ECC and exceed their correction capacity. 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.

IV. Detection of (Un)Jammed Bits

In this section, we develop a novel technique that utilizes physical layer modulation properties to identify (un)jammed bits.

A. Preliminaries on Modulation

I/Q modulation techniques have been widely used in modern wireless systems, including WCDMA, WiMax, ZigBee, WiFi, and DVB (Digital Video Broadcasting). In I/Q modulation, data bits are encoded into physical layer symbols, which are the transmission units in the physical layer. In the following, we take Quadrature Phase-Shift Keying (QPSK) modulation, a typical I/Q modulation, as an example 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 4, 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 symbol received by a receiver is not exactly the same as the original symbol transmitted by the sender, since wireless channels are usually noisy and introduce distortions (e.g., phase and amplitude changes) to signals that pass through them [14]. Hence, in demodulation, the receiver finds the point that is closest to the received symbol in the constellation diagram. For example, in Figure 4, the distance between the received symbol and the point (0, 1) is the minimum, and thus the demodulation output is 00.

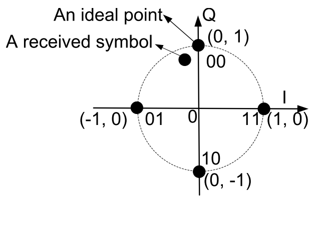

Fig. 4. QPSK modulation/demodulation

B. Observation

Intuitively, jamming signals can introduce a large, even unrecoverable distortion to signals transmitted by the sender, since the goal of the jammer is to interfere with and 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.

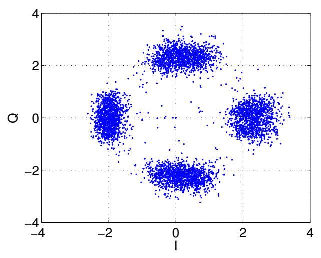

Fig. 5. Normal scenario

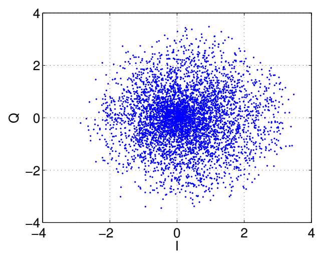

Fig. 6. Jamming scenario

We collect the received symbols using USRPs [21], 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 two communication scenarios: 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. We let the receiver record the coordinates of the received symbols in the constellation diagram.

In the normal scenario, as shown in Figure 5, the received symbols form four clusters, each of which centers around an ideal point of QPSK. However, when there exist jamming attacks, as shown in Figure 6, the received symbols randomly spread over the constellation diagram. It is hard to determine the ideal points for most of the received symbols, and demodulation errors may happen frequently.

C. Detection Method

The above observation provides a possible way to distinguish between jammed and unjammed symbols.

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 5 and 6, 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 then compare d with a threshold t. If $d > t$, then the received symbol is marked as unjammed. Otherwise, it is a jammed symbol and will be discarded.

Note that the detection metric is not the only option. Different metrics can be explored to accommodate different variants of I/Q modulation. For example, rectangular based I/Q modulation (e.g., 64QAM) may use the distance between a received symbol and the closest constellation point as the detection metric. In this paper, 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 in our analysis, this method can enhance the detection accuracy.

Two remaining problems need to be answered: (1) How to determine the detection threshold t, and (2) how well the detection method works. These problems turn out to be related. In the following, we first look at the quality of detection and then develop a method to determine the detection threshold t.

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. In the following, we derive both probabilities of false negative and positive.

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 [2], 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.

Proof: Let (I, Q) and (I_i, Q_i) denote the coordinate of a received symbol and its closest ideal point in a constellation diagram, respectively. Due to jamming, $I \neq I_i$ and $Q \neq Q_i$. For an unjammed symbol, we assume additive white Gaussian noise, and thus I and Q can be represented as $I = I_i + \delta_I$ and $Q = Q_i + \delta_Q$, where δ_I and δ_Q are independent and identically distributed (i.i.d) Gaussian random variables with mean value 0 and variance ${\sigma_N}^2$. According to the properties of Gaussian variables [22], $I = I_i + \delta_I$ and $Q = Q_i + \delta_Q$ are also i.i.d. Gaussian random variables. The mean values of I and Q equal to those of I_i and Q_i, respectively, and the variances of them are all ${\sigma_N}^2$. Let d denote the distance between the received symbol and the origin of the constellation diagram. Thus, $d = \sqrt{I^2 + Q^2}$. According to [26], d follows Rice distribution and the cumulative distribution function $F_d(t)$ of d is $F_d(t) = \mathbb{P}(d \leq t) = 1 - M_1(\frac{v}{\sigma_N}, \frac{t}{\sigma_N})$, where $\mathbb{P}(d \leq t)$ denote the probability that d is less than or equal to t, $v = \sqrt{{I_i}^2 + {Q_i}^2}$, and M_1 is the Marcum Q-function. Note that the probability P_{fp} of false positive equals to the probability that d of at least one symbol in the temporal sequence is less than or equal to t. Thus, $P_{fp} = 1 - (1 - \mathbb{P}(d \leq t))^{2N+1} = 1 - (M_1(\frac{v}{\sigma_N}, \frac{t}{\sigma_N}))^{2N+1}$. □

Theorem 2: (Probability of false negative) Given that each ideal point in the constellation diagram is jammed with equal probability, the probability P_{fn} that a jammed symbol is wrongly 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.

Proof: Let (I, Q) denote the coordinate of a received symbol. I and Q can be represented as $I = I_s + I_j + \delta_I$ and $Q = Q_s + Q_j + \delta_Q$, where (I_s, Q_s) and (I_j, Q_j) are the symbols transmitted by the sender and the jammer, respectively, and δ_I and δ_Q are additive white Gaussian noise. Assume that the sender transmits each ideal point in the constellation diagram with equal probability. Hence, I_s, Q_s, I_j, Q_j are i.i.d random variables. According to central limit theorem, the probability distribution of the average of i.i.d random variables converges to Gaussian distribution as the number of random variables increases. Therefore, we use Gaussian distribution to approximate the distribution of $(I_s + I_j)/2$ and $(Q_s + Q_j)/2$. According to the properties of Gaussian variables [22], $I = I_s + I_j + \delta_I$ and $Q = Q_s + Q_j + \delta_Q$ are also approximately Gaussian distributed, where δ_I and δ_Q are i.i.d Gaussian random variables with mean 0. Ideal points center around the origin, and thus the mean values of I_s, Q_s, I_j, and Q_j are 0.

Let d denote the distance between the received symbol and the origin of the constellation diagram. Thus, $d = \sqrt{I^2 + Q^2}$. Assume that I and Q have the same variance, which is denoted by σ^2. According to [14], d follows Rayleigh distribution and the cumulative distribution function $F_d(t)$ of d is $F_d(t) = \mathbb{P}(d \leq t) = 1 - e^{\frac{-t^2}{2\sigma^2}}$. The probability P_{fn} of false negative equals to the probability that d of all symbols in the temporal sequence are larger than t. Therefore, $P_{fn} = (1 - \mathbb{P}(d \leq t))^{2N+1} = (e^{\frac{-t^2}{2\sigma^2}})^{2N+1}$. □

Experimental Validation: To verify the theoretical probabilities of false positive and false negative, we let the threshold t range between 0 and 5, and for each value of t we run the temporal check enhanced method to detect unjammed symbols from symbols collected for normal and jamming scenarios in our earlier experiment. Ratios of detected symbol number to total symbol number are used to compute the real measured probability of false positive and false negative. (i.e., $P_{fp} = 1 - \frac{\text{\# detected symbols}}{\text{\# total symbols}}$ and $P_{fn} = \frac{\text{\# detected symbols}}{\text{\# total symbols}}$).

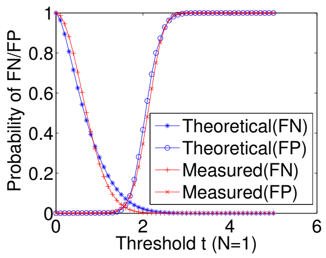

Fig. 7. Theoretical and measured probabilities of false positive/negative when $N = 1$.

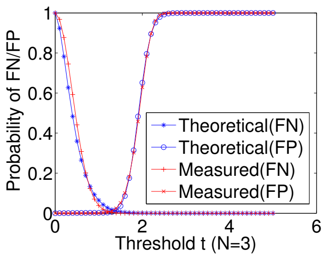

Fig. 8. Theoretical and measured probabilities of false positive/negative when $N = 3$

Figures 7 and 8 show the results for $N = 1$ and $N = 3$, respectively. Meanwhile, we compute P_{fp} and P_{fn} using Theorems 1 and 2. The computation results are also shown in Figures 7 and 8. 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 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 7 and 8, the minimization result and the corresponding t form the intersection point of the P_{fp} and P_{fn} curves.

V. 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 9, 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 V-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 9.

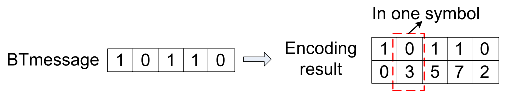

Fig. 9. BitTrickle encoding

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 10 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 this process on all the following symbols until the last symbol is transmitted. Due to jamming and retransmissions, a symbol may get lost or duplicated. In Figure 10, 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 physical layer authenticator.

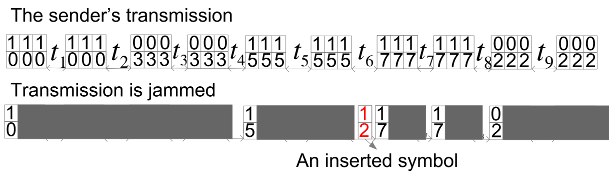

Fig. 10. Transmission Errors

BitTrickle Decoding: The receiver demodulates each received symbol to extract the bit and corresponding label carried by this symbol. Figure 11 shows an example following Figure 10. The extracted bits and labels are 111110 and 052772, respectively. The receiver then takes two steps to correct synchronization errors.

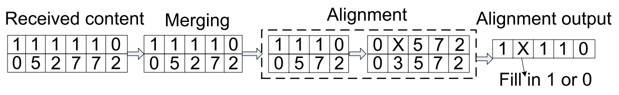

Fig. 11. BitTrickle decoding

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 11, 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 V-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 total number of retransmissions of each bit 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

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

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 reduce decoding failures and filter out inserted bits, we perform alignment on the most correlated part between the positioning code and the merged received labels. We first find the largest common subsequence (LCS) between the positioning code and received labels, and then align the LCS with the positioning code. For example, in Figure 11, the received labels are $05\underline{2}72$, where the underlined label 2 is the inserted label that is from the jammer. The LCS between $05\underline{2}72$ and the positioning code 03572 is 0572. Therefore, this inserted label is filtered.

Note that the 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 [36]. Therefore, we utilize existing dynamic programming method [36] to only find one LCS. It is possible that there exist multiple LCSs and the LCS returned by dynamic programming contain inserted labels[2]. However, such probability decreases quickly with the increase of the percentage of the sender's labels (See Appendix A). 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, given a 0.1 probability that a LCS contains inserted bits. If the sender transmits a BTmessge for 3 times, 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 11. The alignment output is further processed by traditional ECC to recover the original message. There may exist multiple alignment outputs, since the merged labels may fit multiple combinations of positions in the positioning code. In Section V-C, we develop a fast alignment approach that not only achieves desired alignment accuracy, but reduces the overhead by only trying a subset of all combinations.

Diversity Degree of a Positioning Code: Note that consecutive bits of a BTmessage may happen to be the same, and they will be treated as duplication of a single bit and incorrectly merged together if they have the same label. To avoid such situation, 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

In the encoding, 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 not exceed $2^{\log_2 M - 1} - 1 = \frac{M}{2} - 1$. 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 also use backoff time between transmitting two consecutive symbols of an MDC to reduce the chance of colliding with the jammer's signals.

The receiver then demodulates the symbols of the received BTmessage, and extracts a data bit and a label from each symbol. The number of extracted data bits equals the number of symbols. The receiver 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

[2]For example, consider the positioning code 66433 and received labels $4\underline{6}3\underline{6}3$, where underlined labels are inserted labels. The LCSs between the positioning code and received labels are $\underline{66}3$ and 433. The receiver cannot tell which one is "good".

together and points the cursor to the next bit/label. The receiver repeats the same process until all bits/labels of the received BTmessage are scanned. The expected number of comparisons is $\frac{L}{(N_r-1)/2} \times \frac{N_r-1}{2} = L$, where L is the message length.

Merging Errors: Different bits may be incorrectly merged together. Lemma 1 and Theorem 3 give the upper bound of the probability of merging errors.

Lemma 1: The probability that two labels in a positioning code equal to each other is less than or equal to $\frac{1}{R-d}$, where d is the diversity degree of the positioning code and R is the number of possible values for each label (e.g., R=4 for 8PSK).

Proof: Let $S = s_1||...||s_n$ denote the positioning code, where n is the number of labels in it. Let p_{eq} denote the probability that the i-th label s_i equals to the j-th label s_j, where $1 \le i, j \le n$ and $i \ne j$. Without loss of generality, we assume that $j > i$. According to the diversity requirement of a positioning code (See Section V-A), $s_j \ne s_{j-1}, ..., s_{j-d}$ and $s_i \ne s_{i+1}, ..., s_{i+d}$. If $j - i \le d$, s_j is one of the previous d labels of s_i. Hence, $p_{eq} = 0$. If $j - i = d + 1$, s_j's previous d labels are actually s_i's successive d labels, and thus $s_j \ne s_{j-1}, ..., s_{j-d}$ and $s_i \ne s_{j-1}, ..., s_{j-d}$. Therefore, $p_e = \frac{1}{R-d}$. If $d + 1 < j - i \le 2d$, there exists an overlap between s_j's previous d labels and s_i's successive d labels, but the length of the overlap is less than d. Thus, $p_e < \frac{1}{R-d}$. If $j - i > 2d$, there is no overlap and $p_e = \frac{1}{R}$. Thus, p_e is at most $\frac{1}{R-d}$. □

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 by the sender is lost.

Proof: Let $S = s_1||...||s_n$ and $M = m_1||...||m_n$ denote the positioning code and original BTmessage, respectively. Let $M_r = m_{i_1}^{r_1}||...||m_{i_g}^{r_g}$ denote the received BTmessage, where m_{i_j} is the i_j-th element of the original BTmessage and $m_{i_j}^{r_j}$ means that the receiver receives r_j retransmitted copies of m_{i_j}. $m_{i_j}^{r_j}$ may be incorrectly merged with $m_{i_{j+1}}^{r_{j+1}}$ if they are identical and have the same label (i.e., $m_{i_j} = m_{i_{j+1}}$ and $s_{i_j} = s_{i_{j+1}}$).

An information bit (i.e., a bit in the original BTmessage and all its retransmitted copies) may get lost. Let p denote the probability that a transmitted bit is lost. The probability that an information bit is lost equals to p^r, where r is the number of retransmissions. If $i_{j+1} - i_j \le d$ (i.e., less than d information bits between $m_{i_{j+1}}$ and m_{i_j} are lost), then $m_{i_{j+1}}$ is among the successive d elements of m_{i_j} and their labels are always different. Thus, the probability of merging errors is 0. If $i_{j+1} - i_j > d$, $m_{i_j}^{r_j}$ and $m_{i_{j+1}}^{r_{j+1}}$ will be identified as duplicate bits when $m_{i_j} = m_{i_{j+1}}$ and $s_{i_j} = s_{i_{j+1}}$. Let p_{eq} be the probability that the i-th element s_i equals to the j-th element s_j, where $1 \le i, j \le n$ and $i \ne j$. The overall probability p_{e_j} that $m_{i_j}^{r_j}$ is incorrectly merged with $m_{i_{j+1}}^{r_{j+1}}$ is $\frac{p_{eq}}{2}\mathbb{P}(i_{j+1} - i_j > d) = \frac{p_{eq}}{2}\sum_{k=d}^{n-n(1-p^r)} p^{rk}(1-p^r)$. According to Lemma 1, $p_{eq} \le \frac{1}{(R-d)}$. Therefore, $p_{e_j} \le \frac{(p^{rd}-p^{r(n-n(1-p^r)+1)})}{2(R-d)}$, where $n(1-p^r)$ is the expected number of information bits received by the receiver. For the received BTmessage $M_r = m_{i_1}^{r_1}||...||m_{i_g}^{r_g}$, the probability p_e of merging incorrectly is 1-$\prod_{j=1}^{j=n(1-p^r)-1}(1 - p_{e_j})$, and thus $p_e \le 1 - (1 - \frac{p^{rd}-p^{r(n-n(1-p^r)+1)}}{2(R-d)})^{n(1-p^r)-1}$. □

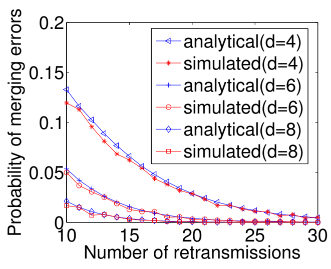

Fig. 12. Simulated probability of merging errors and analytical upper bound.

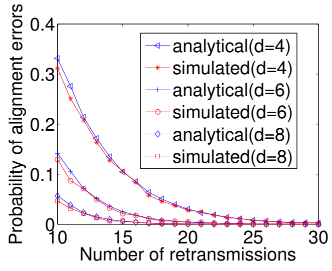

Fig. 13. Simulated probability of alignment errors and analytical upper bound.

We use simulation to validate the above analytical upper bound.[3] We let $R = 32$ and $p = 0.95$, and perform 10,000 trails in our simulation. In each trial, we randomly generate a message and a positioning code whose length is 155, and label the message using the positioning code. We retransmit each bit of the message for r times ($10 \le r \le 30$), 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 12, the simulated probability of merging error is only slightly less than its analytical upper bound, which indicates that the upper bound computed by Theorem 3 is a tight upper bound. It shows that 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 11, the merged labels are 0572). As discussed earlier, the receiver can feed L into S to determine the positions of received bits. For example, if $L = 17$ and $S = 1317$, either the first and the last bit or the last two bits of the original BTmessage are received.

[3]Simulations are done in MATLAB 7.7.0 on a computer with a 2.30 GHz AMD CPU and 4.0 GB memory

(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. The number of comparisons the above method requires is $\binom{n}{q}$, where n is the length of S.

If n is large, the brute force 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 instead of all possible ones in a brute force way. 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. Without loss of generality, we assume that the fast alignment finds the leftmost subsequence of the positioning code. In Theorem 4, we derive an upper bound for the probability of alignment errors.

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 transmitted by the sender is lost.

Proof: Let $S = s_1||...||s_n$ denote the sequence formed by the positioning code and p_{eq} be the probability that the i-th element s_i equals to the j-th element s_j, where $1 \leq i, j \leq n$ and $i \neq j$. According to Lemma 1, $p_{eq} \leq \frac{1}{R-d}$, where R and d are the number of labels and the diversity degree of the positioning code, respectively. Let $F = f_1||...||f_q$ denote the sequence formed by the actual positions of received bits (i.e., the receiver receives the f_1-th,...,f_q-th bits), and L denote the sequence formed by merged labels. F is the positions of S's leftmost subsequence that equals L if two conditions are satisfied: (1) For $1 \leq i < f_1$, $s_i \neq s_{f_1}$. (2) For $1 \leq j \leq q-1$ and $f_j < i < f_{j+1}$, $s_i \neq s_{f_{j+1}}$.

Therefore, the probability p_{min} that F is the positions of S's leftmost subsequence is $\prod_{i=1}^{f_1-1}(1 - \mathbb{P}(s_i = s_{f_1}))\prod_{j=1}^{q-1}\prod_{i=f_j+1}^{f_{j+1}-1}(1 - \mathbb{P}(s_i = s_{f_{j+1}}))$. Assume that $f_j < i < f_{j+1}$. Thus, $\mathbb{P}(s_i = s_{f_{j+1}}) = p_{eq}\mathbb{P}(f_{j+1} - i > d)$. According to Lemma 1, $p_{eq} \leq \frac{1}{(R-d)}$. Thus, $\mathbb{P}(s_i = s_{f_{j+1}}) \leq \frac{\mathbb{P}(f_{j+1}-f_j>d)}{R-d}$. Note that $f_{j+1} - f_j > d$ indicates that at least d labels between s_{f_j} and $s_{f_{j+1}}$ are lost. Therefore, $\mathbb{P}(s_i = s_{f_{j+1}}) \leq \frac{p^{rd}-p^{r(n-q+1)}}{R-d}$ Similarly, for $1 \leq i \leq f_1$, $\mathbb{P}(s_i = s_{f_1}) \leq \frac{p^{rd}-p^{r(n-q+1)}}{R-d}$ Thus, $p_{min} \geq (1 - \frac{p^{rd}-p^{r(n-q+1)}}{R-d})^{f_q-q}$, where f_q is a random variable ranging from q to n. According to total probability formula, $p_{min} \geq \sum_{k=q}^{n} \frac{\binom{k}{q}}{\sum_{w=q}^{n}\binom{w}{q}}(1 - \frac{p^{rd}-p^{r(n-q+1)}}{R-d})^{k-q}$. The probability p_e that the alignment is incorrect equals to the probability that F is not the positions of S's leftmost subsequence. Hence, $p_e = (1-p_{min}) \leq 1-\sum_{k=q}^{n} \frac{\binom{k}{q}}{\sum_{w=q}^{n}\binom{w}{q}}(1-\frac{p^{rd}-p^{r(n-q+1)}}{R-d})^{k-q}$. $\square$

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 of the positioning code 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 13 shows that the simulated probability of alignment error and the analytical upper bound decrease as the number of retransmissions increases, and a larger diversity degree d can lead to a smaller error probability. The upper bound computed by Theorem 4 is a tight upper bound of the error probability. In particular, when $d = 8$ and $r = 20$, both the simulated probability and the analytical upper bound are about 0.0006.

VI. 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].

Communication Flow: As shown in Figure 14, The sender encodes a message using the BitTrickle encoder, and then modulates the binary bits of the encoded message into symbols. Each symbol is further split into in-phase (I) and quadrature-phase (Q) components, and the sender multiplies I component and Q component by a cosine and sine carrier signal, respectively. The outcomes of both multiplication are superimposed, resulting in the modulated signal. Finally, the sender uses a D/A converter to transform the modulated signal into RF signal and transmits.

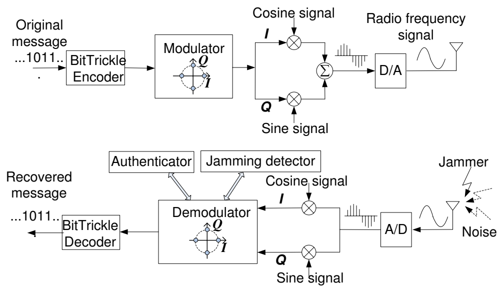

Fig. 14. A Communication Framework of BitTrickle

In the reception process, the receiver applies an A/D sampler to transform a RF signal into a modulated signal, and then multiplies the digital modulated signal by cosine and sine carrier signals to get the I and Q components, which together represent a physical layer symbol. Each symbol enters a demodulator that can decode symbols into binary bits. A jamming detector and an authenticator run on top of the demodulator. They decide whether a symbol is jammed or fake by looking at intermediate demodulation results, and feed their decisions to the demodulator, which will discard all "bad" symbols. Finally, the receiver uses the BitTrickle decoder to process the demodulation output and recover the original message.

Reactive Jammer: A reactive jammer senses the channel and transmits jamming signals once it detects a sender's signal. 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).

Note that a reactive jammer not only transmits jamming signals, but receives from the wireless channel to detect legitimate user's transmission. 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 reactive 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 the directory gnuradio/gnuradio-examples/python/digital.
3) **802.11 DSSS**–IEEE 802.11 protocol running at direct-sequence spread spectrum (DSSS) mode on 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, error-free 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 use the temporal based detection method discussed in Section IV-C to detect jammed symbols. To examine the the performance of jamming detector in terms of false negative rate and false positive rate, we let the receiver collect the distance of each symbol from the origin of the constellation diagram and perform off-line analysis in MATLAB. Figure 15 shows the result for temporal sequence length $N = 5$. We can see that a threshold of 0.3 balances the false negative and false positive. In our implementation, we set the threshold and N to be 0.3 and 5.

Physical Layer Authenticator: We develop a simple device authenticator based on [7], 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),

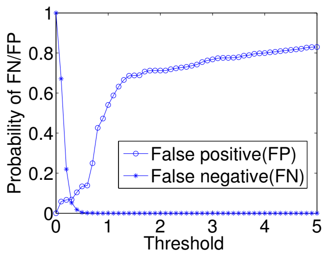

Fig. 15. False negative and positive of jamming detector

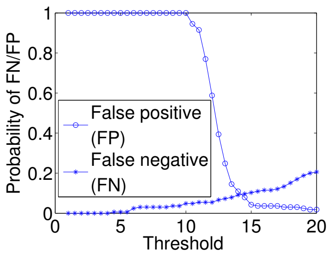

Fig. 16. False negative and positive of authenticator

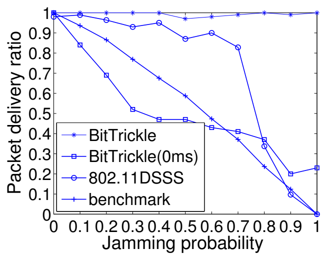

Fig. 17. Packet delivery ratio v.s. jamming probability.

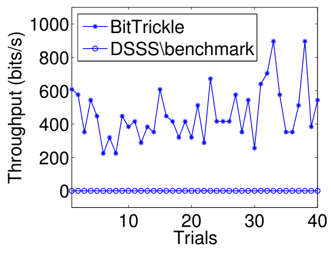

Fig. 18. Throughput when jamming probability is 1.

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 collect EVMs of the sender and the jammer and perform off-line analysis in MATLAB. Figure 16 shows the result. For a threshold that equals to 14.5, the false negative and the false positive rate achieved by the authenticator are 0.0970 and 0.0788, respectively. Noted that we use a very simple physical layer authenticator in our prototype system. Certainly other advanced authenticators can be adopted to get an even lower false rate and higher security.

B. Performance of BitTrickle

We set the transmission 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 a single packet 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: To examine the performance, we consider different jamming intensities. We use a probabilistic reactive jammer that jams at probability p for $0 \le p \le 1$ 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 17 shows the result.

(1) 802.11 DSSS and GNURadio Benchmark: Packet delivery ratio decreases as jamming probability increases. Note that GNURadio benchmark does not employ ECC and retransmission mechanism, and thus the packet delivery ratio decreases at a rate linearly proportional to the jamming probability. 802.11 DSSS uses packet retransmissions and forward error correction. Hence, it achieves a higher packet delivery ratio than GNURadio benchmark. However, when jamming probability exceeds 0.7, the performance of 802.11 DSSS degrades dramatically. For both 802.11 and GNURadio benchmark, when the jamming probability equal to 1 (i.e., the jammer always jams when it senses a transmission), the packet delivery ratio drops to 0, and no data packets can be delivered.

(2) BitTrickle: For BitTrickle, we let the sender takes a random backoff ranging between 150–200 ms after it transmits every 6 bits, each bit of which is retransmitted for 15 times. Figure 17 shows that BitTrickle achieves a stable packet delivery ratio that fluctuates around 1 no matter how jamming probability varies.

We then reduce the backoff time to 0 ms and increase the bit retransmissions to 60. Figure 17 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 jamming probability is 1, unlike GNURadio benchmark and 802.11 DSSS, BitTrickle with zero backoff still achieves a non-zero packet delivery ratio (i.e., more than 0.2).

Throughput: We consider a common jamming scenario, where the reactive jammer jams the channel as long as it hears the target signal(i.e., $p = 1$). The backoff time of BitTrickle is set to be 0 ms. We performs 40 trials. In each trial, the sender transmits 100 data packets to the receiver and we compute throughput as $\frac{\text{\# correct packets (blocks)} \times \text{packet (block) length}}{\text{transmission time}}$. Figure 18 plots the computed throughput for each trial. The GNURadio benchmark and 802.11 DSSS fail to deliver any packet, whereas BitTrickle still achieves a throughput that ranges between 200–900 bits/s, allowing wireless communication to continue

To understand how ECC coding rate affects the throughput, we test the BitTrickle throughput using RS(155,55) and RS(60,36), respectively. Figure 19 plots the throughput as a function of signal-to-jamming ratio (SJR), the ratio of the reaction time to jamming duration. Figure 19 shows that RS(60,36) leads to a higher throughput than RS(155,55) when SJR is less than 0.25. The reason is that RS(60,36) requires a shorter positioning code than RS(155,55), which reduces the chance of synchronization errors caused by alignment. As SJR increases, the receiver can get more information from the sender, and thus the probability of synchronization errors decreases. Note that the error correction capability of RS(155,55) is stronger than that of RS(60,36). For small SJRs,

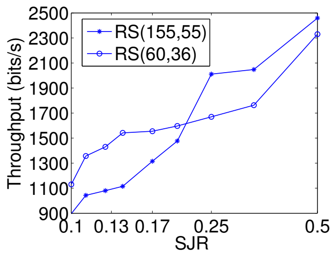

Fig. 19. Throughput for different coding rate.

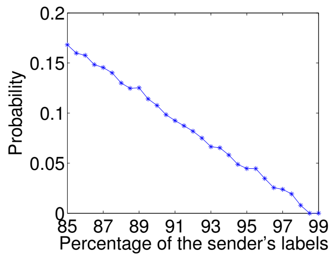

Fig. 20. Probability of containing fake labels.

RS(155,55) does not suffer from severe synchronization errors, and thus it can correct more substitution errors and achieve a better throughput. When attacked by a jammer with SJR = 0.5, the throughput of the prototype system using RS(155,55) is about 2.5kbps. Figure 19 indicates that we can improve the throughput of BitTrickle by choosing an appropriate coding rate for different SJRs.

Remarks: The experiments reveal the following: (1) It is possible that a reactive jammer can defeat traditional approaches if the jammer is broadband and has high transmit power. As an example shown in the experiments, the throughput of 802.11 DSSS is zero and no packets can be delivered when it is attacked by the reactive jammer; (2) Even if traditional methods fail, the experiments indicate that BitTrickle can still allow wireless nodes to establish communication in the presence of a powerful reactive jammer by taking advantage of the channel sensing behavior of reactive jammers; (3) The efficiency of BitTrickle can be improved by adjusting system parameters (e.g., choosing an appropriate coding rate).

VII. Related Work

FHSS and DSSS (e.g., [6], [14], [15], [20], [24], [27]–[29], [31], [32], [34], [37]) 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 [40]. The idea is to map the inter-arrival times of a sender's corrupted packets into information bits [18]. However, 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 [30]. The jamming detector in [30] 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. There exist other related work, including methods for identifying insider jammers [9], [10], mitigating jamming of control channels [18], [33], jamming avoidance and evasion [5], [39], [42], and mitigating jamming in sensor networks [19], [39]. To the best of our knowledge, this is the first work that addresses both broadband and high power reactive jamming.

VIII. 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.

References

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

[2] Marcum q-function. http://mathworld.wolfram.com/MarcumQ-Function.html.

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

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

[5] L. Baird, W. Bahn, and M. Collins. Jam-resistant communication without shared secrets through the use of concurrent codes. Technical report, US Air Force Academy, 2007.

[6] A. J. Berni and W. D. Greeg. On the utility of chirp modulation for digital signaling. *IEEE Transaction on Communications*, 21:748–751, 1973.

[7] 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.

[8] 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.

[9] J. Chiang and Y. Hu. Extended abstract: Cross-layer jamming detection and mitigation in wireless broadcast networks. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '07)*, 2007.

[10] J. Chiang and Y. Hu. Dynamic jamming mitigation for wireless boradcast networks. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM '08)*, 2008.

[11] H.C. Choe, C.E. Poole, A.M. Yu, and H.H. Szu. Novel identification of intercepted signals from unknown radio transmitters. In *SPIE 2491, 504 (1995)*, 1995.

[12] L. H. A. Correia, D. F. Macedo, D. A. C. Silva, A. L. dos Santos, A. A. F. Loureiro, and J. M. S. Nogueira. Transmission power control in mac protocols for wireless sensor networks. In *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 282–289, New York, NY, USA, 2005. ACM.

[13] 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.

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

[15] S. Hengstler, D. P. Kasilingam, and A. H. Costa. A novel chirp modulation spread spectrum technique for multiple access. In *Proceedings of the IEEE International Symposium on Spread Spectrum Techniques and Applications*, pages 73–77, September 2002.

[16] 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, New York, NY, USA, 2008. ACM.

[17] S. Koskie and Z. Gajic. A nash game algorithm for sir-based power control in 3g wireless cdma networks. *IEEE/ACM Transaction on networking*, 13(5):1017–1026, 2005.

[18] 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.

[19] M. Li, I. Koutsopoulos, and R. Poovendran. Optimal jamming attacks and network defense policies in wireless sensor networks. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM '07)*, 2007.

[20] 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.

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

[22] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. Mc-Graw Hill, 1984.

[23] 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.

[24] C. Pöpper, M. Strasser, and S. Čapkun. Jamming-resistant broadcast communication without shared keys. In *Proceedings of the USENIX Security Symposium*, 2009.

[25] M. Rasti, A. Sharafat, and B. Seyfe. Pareto-efficient and goal-driven power control in wireless networks: a game-theoretic approach with a novel pricing scheme. *IEEE/ACM Transaction on networking*, 17(2):556–569, 2009.

[26] S. O. Rice. Mathematical analysis of random noise. *Bell System Technical Journal*, 24:46–156, 1945.

[27] R. A. Scholtz. Multiple access with time hopping impulse modulation. In *Proceedings of the 2008 IEEE MILCOM conference*, pages 447–450, 1993.

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

[29] A. Springer, W. Gugler, M. Huemer, L. Reindl, C. C. W. Ruppel, and R. Weigel. Spread spectrum communications using chirp signals. In *Proceedings of the IEEE/AFCEA EUROCOMM Conference*, pages 166–170, May 2000.

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

[31] 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*, pages 64–78, 2008.

[32] M. Strasser, C. Pöpper, and S. Čapkun. Efficient uncoordinated FHSS anti-jamming communication. In *Procceedings of MobiHoc '09*, May 2009.

[33] P. Tague, M. Li, and R. Poovendran. Probabilistic mitigation of control channel jamming via random key distribution. In *Proceedings of IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '07)*, pages 1–5, 2007.

[34] D. Torrieri. *Principles of Spread-Spectrum Communication Systems*. Springer, 2004.

[35] X. Wang, M. Fei, and X. Li. Performance of chirp spread spectrum in wireless communication systems. In *Proceedings of the 11th IEEE Singapore International Conference on Communication Systems*, pages 466–469, Novemeber 2008.

[36] Wikipedia. Longest common subsequence problem, 2011. [Online; accessed 10-Feb-2011].

[37] M. Win and R. Scholtz. Impulse radio: How it works. *IEEE Communications Letters*, 2(2):36–38, February 1998.

[38] W. Xu, K. Ma, W. Trappe, and Y. Zhang. Jamming sensor networks: Attack and defense strategies. *IEEE Network*, pages 41–47, 2006.

[39] W. Xu, W. Trappe, and Y. Zhang. Channel surfing: Defending wireless sensor networks from jamming and interference. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, 2007.

[40] 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.

[41] 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.

[42] W. Xu, T. Wood, W. Trappe, and Y. Zhang. Channel surfing and spatial retreats: Defenses against wireless denial of service. In *Proceedings of the 3rd ACM Workshop on Wireless Security (WiSe '04)*, 2004.

[43] J. Zhang, M. H. Firooz, N. Patwari, and S. K. Kasera. Advancing wireless link signatures for location distinction. In *MobiCom '08: Proceedings of the 14th ACM international conference on Mobile computing and networking*, New York, NY, USA, 2008. ACM.

Appendix

We examine the relationship between the percentage of the sender's labels and the probability that a LCS still contains inserted labels through computer simulations. We perform 10,000 trails, and let jamming probability and the number of labels be 0.95 and 64 respectively. In each trial, we randomly generate a positioning code of length 60, jam each label with the jamming probability, and insert fake labels into the remaining labels. The outcome are merged together. Then we find the LCS between the merged outcome and the positioning code. If the LCS contains fake labels, we increase a counter by 1. The probability that inserted labels exist in the LCS is calculated as the ratio of the counter value to 10,000. Figure 20 shows that high percentage of the sender's labels lead to low probability that the LCS contains fake labels. In particular, when the sender's labels account for more than 95% of total received labels, this probability can be lower than 0.05.