Mitigating Constant Jamming in Cognitive Radio Networks using Hybrid FEC Code

Victor Balogun
Computer Science Department
University of Idaho
Moscow, ID, U.S.A.
Email:balo8072@vandals.uidaho.edul

Axel Krings
Computer Science Department
University of Idaho
Moscow, ID, U.S.A.
Email:krings@uidaho.edu

***Abstract*—The task of detecting and mitigating Cognitive Radios (CRs) operating as constant jammers in a Cognitive Radio network (CRN) can be very daunting. The CR constant jammers prey on the adaptable functionalities of CRN so as to cause serious denial of service to the users of the network. In addition, these jammers are capable of introducing value faults in pathological cases as a result of being able to manipulate transmitted data. In previous research, we have investigated the performance of CRNs operating in the presence of jamming attacks that are capable of introducing value faults. The results obtained show that CR constant jammers are very effective in their operations and they are capable of bringing down the entire CRN when their jamming rate is just about 30%. In this paper, we show that none of the existing anti-jamming solutions are able to mitigate CR jammers that are capable of manipulating communicated data. As a result, we propose a hybrid forward error correction (FEC) code that incorporates data integrity checking into an efficient forward error correction mechanism. The approach uses data redundancy to remove the need for retransmission of lost or detected manipulated data caused by jamming attacks by exploring the recovery block component of the proposed solution. We present the algorithm for our proposed solution and evaluate its performance through simulation. The result of our analysis using suitable performance metrics shows that the solution is very efficient and robust against the different rates of jamming perpetrated by constant jammers.**

I. Introduction and Background

The Cognitive Radio Network (CRN) has been adopted as a technology that will alleviate the spectrum shortage problem [1]. The technology allows unlicensed users to share the unused part of the licensed bands of spectrum with the incumbents when present (*Underlay approach*) or during the incumbents' OFF period (*Overlay approach*). The benefits of CRNs are broad and range from providing low cost Internet connectivity to providing easy access to government-delivered services to rural and underserved regions. Accurate detection of the presence of the incumbents by CRs is a crucial issue as the CRN standard does not tolerate any level of interference to the incumbent's network.

Channel impairments like deep shadowing [16] and multi-path fading [14] are some of the barriers to adequate sensing of the spectrum for the presence of an incumbent by a CR. Therefore, a CR needs the assistance of other CRs operating in its neighborhood to be able to accurately sense the spectrum. This is called cooperative spectrum sensing (CSS). Different CSS architectures were proposed in [2] and [3]. Despite the expected success and potentials of CSS, the presence of jammers operating as CRs in a CSS Cognitive Radio network is of great security concern. This is because the CR jammers have reconfigurable features that make them capable of introducing value faults along the transmission channels. The legitimate CR nodes themselves are easy prey to such attackers as they could be manipulated by their reconfigurable features so as to cause serious DoS to the entire network. In order to represent CRN as a network that is capable of exhibiting different types of faults including value faults, the research in [3] presents different jamming scenarios in a CRN based on a hybrid fault model that identifies the possibility of different fault types including omissive and transmissive value faults. The impact of these jammers have been investigated in [4] and the effect of CR jamming, especially the case of constant jammers on the average throughput of the network is shown to be huge. Therefore, the need to design a strategy that is capable of mitigating this category of CR jammers is of great importance. To the best of our knowledge, none of the existing anti-jamming mechanism is capable of effectively mitigating CR jammers that are able to manipulate transmitted data in CRNs.

Based on the hybrid fault model classification in [3] and the perceived impacts of jamming on this model as quantified in [4], the main contribution of this paper is a new hybrid forward error correction (FEC) code that is capable of mitigating jamming attacks under this classification. Furthermore, we present the algorithmic design of the proposed solution, its implementation in Network Simulator 2 (NS-2) and the analysis of its performance measurement using suitable metrics.

1) Anti-jamming Strategies for CR Networks : Several authors [5], [6] have investigated jamming in Cognitive Radio Networks. Some of these authors have proposed the use of Spread Spectrum Frequency Hopping (FH) and Direct Sequence Spread Spectrum (DSSS) as a solution to alleviate jamming attacks in wireless networks. The popular approach is to spread the signal over a larger bandwidth, thereby making it costly for jammers to hinder an on-going transmission. The combination of Spread Spectrum and Orthogonal Frequency Division Multiplexing [7] was also considered as an efficient means of mitigating jamming attacks in wireless networks. In CRNs based on the Spread Spectrum approach [7], the available spectrum is divided into several pieces of non-overlapping channels in which only a small portion of the channels is used for transmission at a time. The malicious jammer either jams a large number of the channels with negligible jamming

effect in each channel or jams few channels, which might not be in use by the CRs. The problem with the Spread Spectrum approach is that a CR jammer is assumed to have knowledge of the hopping sequence and therefore could adapt to the hopping sequence of other CR nodes. Whereas most research has addressed jamming resulting in benign faults, our assumption is to include malicious faults.

2) Forward Error Correction (FEC) Codes: FEC schemes like LT (Luby Transform) code [9], Raptor code [11], and Low Density Parity-Check Code (LDPC) [10], can be used to recreate data lost due to jamming attacks in a CRN through data redundancy. The Raptor code is a type of Fountain code [10] with linear time encoding and decoding, in which a message made up of a number of k symbols is encoded into an infinite series of symbols in such a way that if during transmission some part of the data is lost, e.g., due to jamming, the lost data can be recovered with a non-zero probability that increases as the number of the received symbols increases beyond k [11]. The shortcoming of an FEC code like the Raptor code is that it has not been investigated for CRNs Furthermore, the Raptor code can only deal with benign and omissive faults.

II. Hybrid FEC Code Formulation

In order to overcome the limitations identified in the previous section, we propose a hybrid FEC code defined by the concatenation of the Raptor code and the Secure Hash Algorithm-2 (SHA-2). SHA-2 [12] is the common name used for three types of computationally efficient hash functions, namely SHA-256, SHA-384, and SHA-512, that transform any set of data elements into 256 bit, 384 bit and 512 bit fixed length values respectively. The output of SHA-2, known as message digest, can then be used to verify the integrity of the original data sent against the message digest of the data received at the destination.

We therefore propose to use the Raptor code part of the hybrid code to recover any data loss due to omissive faults as a result of jamming and SHA-2 part to handle transmissive (value) faults due to jamming. Any value fault due to malicious jamming or a bad channel will be detected with SHA-2 if the message digest generated at the receiver is different from the message digest generated at the sender. This is because it is computationally infeasible to find two different messages having the same message digest or to find a message that hashes to a given message digest. Detection of value faults can result in different actions, depending on the CRN application scenario. We therefore identify these scenarios as the recovery block [17] for the hybrid code, which is discussed next.

A. Recovery Block

Whenever the hybrid code detects a discrepancy between the message digest of the data sent and the message digest of the data received, the line of action to be taken could be any of the options listed below depending on the application scenario:

1) The Raptor part of the code can be used to iteratively correct suspected corrupted bits of the message as if it was omitted and SHA-2 is used to regenerate the message digest each time to verify the data received. The iteration process involves extending the number of encoded symbols received at the destination beyond k so as to reduce the probability of a decoding failure of the Raptor code. After a suitable number of iterations the received encoded symbols are decoded and an output message is generated for the receiver with the Raptor code. The SHA-2 is again used to compute the message digest of the received message, which is compared with that of the original sent message.

2) The erroneous message is discarded and the system is suspected to be under jamming attack, requiring further actions, (i) e.g., moving the CR nodes away from the suspected source(s) of jamming, (ii) re-authenticating all the CR nodes participating in the CRN activities, (iii) re-validating the activities of these CR nodes to see if there is any violation of any of the CR security requirements, and (iv) isolating any suspected jamming nodes from the CRN.

3) If the channel experiencing jamming is a CR control channel, the CRN with its adaptable capability switches to another channel that is not experiencing jamming at that instance. It then routes control messages through this channel, thereby making it the new control channel.

4) If all the earlier mentioned options fail, the receiver requests retransmission of the message as a last resort.

B. Design of the Hybrid Code

The design of the proposed hybrid FEC code is illustrated with Figure 1, showing the flowchart of the complete process. The proposed hybrid FEC code operates between the application layer and the transport layer. Before a sender's message is sent from the application layer to the transport layer, the SHA-2 module is used to generate the message digest. This message digest is inserted into the message's packet header and the message is passed to the encoder part of the Raptor module. At the precode stage of the encoder, redundant symbols are added to the message while at the LT code stage, the LT code is used to generate the encoded output symbols. These encoded output symbols are passed down to lower layers of the protocol stack. At the receiver, when the message arrives at the transport layer, the Raptor decoder starts the decoding process as soon as it receives about $k(1+\varepsilon)$ of the encoded output symbols. Once the decoding process is successful, the decoded message is sent to the SHA-2 module where the message digest of the decoded message is generated. The message digest generated is then compared with the message digest stored in the packet's header of the original message. If the two message digests match, then the message is received with no error and forwarded to the receiver through the application layer. If there is a difference between these two message digests, then it means that the message has been manipulated along the transmission path. The hybrid code will therefore initiate the recovery block procedure described in sub-section II-A.

III. The Algorithm for the Hybrid Code

Since the proposed hybrid FEC code is made up of the SHA-2 and Raptor code, we first describe the SHA-2 computation process and later describe the Raptor algorithm part of the code.

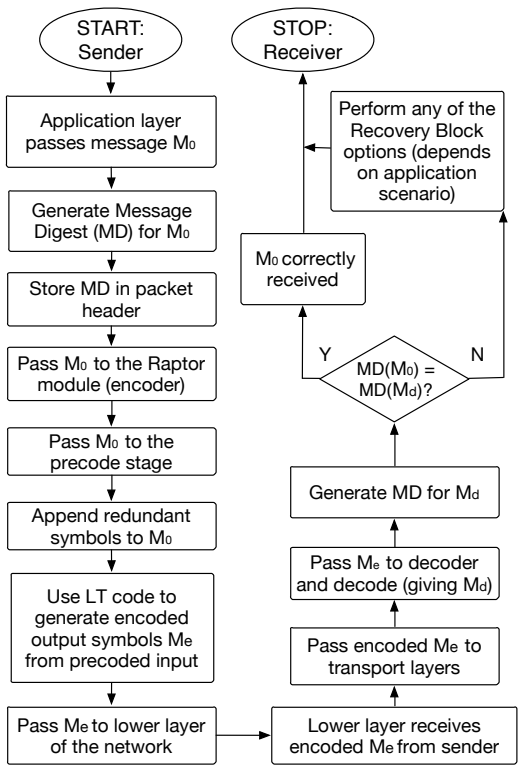

Fig. 1. Flowchart for the Hybrid FEC Algorithm

A. Algorithm for SHA-2

SHA-256, the 256-bit version of SHA-2 was chosen for the implementation of the hybrid FEC code. SHA-256 was defined in the NIST (National Institute of Standards and Technology) standard [13]. A message, M of length l can be hashed with SHA-256 to produce a 256-bit message digest provided $0 \leq l < 2^{64}$. The SHA-256 algorithm is divided into two stages: (i) Preprocessing and (ii) Hash Computation. We here present a summary of the different stages of the algorithm as discussed in [13].

1) Preprocessing: The preprocessing stage involves three essential steps which are:

i) Setting the initial hash value, $H^{(0)}$, which consists of eight 32-bit words in hexadecimal. They are obtained from taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.

ii) Padding the message, M. Padding is done to ensure that the padded message is a multiple of 512 bits. A message, M of length l bits is padded by appending the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $l+1+k \equiv 448 \bmod 512$. Then we append the 64-bit block that is equal to the number l using a binary representation.

iii) Parsing the message into message blocks. The message and its padding are parsed into N 512-bit blocks, which can be expressed as sixteen 32-bit words. The first 32 bits of message block i are denoted $M_0^{(i)}, ..., M_{15}^{(i)}$.

2) SHA-256 Hash Computation: Addition (+) in SHA-256 is performed modulo 2^{32}. A list of constants and functions used in computing the hash is presented in the standard. Each message block, $M^{(1)}, M^{(2)}, ..., M^{(N)}$, is processed in order, using the hash computation procedure described in [13]). After repeating the steps for a total of N times (i.e., after processing $M^{(N)}$), the resulting 256-bit message digest of the message, M, is $H_0^{(N)} H_1^{(N)} H_2^{(N)} H_3^{(N)} H_4^{(N)} H_5^{(N)} H_6^{(N)} H_7^{(N)}$.

B. Algorithm for Raptor Code

We here present a brief description of the Raptor code algorithm. For theoretical details and proof of this algorithm, the reader is referred to [10]. The algorithm of the Raptor code could be formulated in matrix representation in which multiplication operations performed on the matrices yield solutions that are equivalent to executing the Raptor code algorithm [10]. We first present the formal description of the Raptor code algorithm and then describe the matrix interpretation equivalent of the algorithm. The Raptor code is made up of two essential components: (i) Raptor encoder (ii) Raptor decoder.

1) Algorithm for Raptor Encoder: The Raptor encoder is made up of two stages:
Pre-code stage: This stage is usually done with an LDPC code but other codes like Gray code [15] can be used as well. The source or input symbols k are mapped unto l intermediate symbols such that $l < k$.
LT-encoding stage: At this stage, the intermediate symbols from the pre-code stage are used to generate each encoded symbol by carrying out an eXclusive-OR (XOR) operation on a subset of symbols randomly selected from the l intermediate symbols in such a way that it agrees with a certain probability distribution.

Using the matrix interpretation, the Raptor encoder algorithm can be represented as operations that involve performing matrix multiplications with vectors with the intent of solving systems of equations that represent the Raptor encoding process. The coefficients of these matrices are either zero or one and therefore are said to be binary. The vectors are vectors of symbols, with each symbol represented as a binary digit. Using this interpretation, the "pre-code stage" of a Raptor code is defined as the multiplication:

$$u^T = G.x^T \tag{1}$$

where G is an $n \times k$ binary matrix representing the generator matrix of the pre-code and x represents the vector $(x_1, ..., x_k)$ made up of the input vectors.

According to [10], to any given set of N output symbols of the Raptor code, there corresponds a binary $N \times n$-matrix, S. This implies that matrix S contains rows that are independently sampled from the distribution $\Omega(x)$ and they are vectors that corresponds to the output symbols of the Raptor code. The relationship is represented as:

$$S.G.x^T = z^T \tag{2}$$

where $z = (z_1, ..., z_N)$ represents the encoded output symbols of the Raptor code that is made up of the column vector.

2) Algorithm for Raptor Decoder: The decoding algorithm for a Raptor code is defined as an algorithm of length m that can regain k input encoded symbols out of any set of m output symbols and errs with a probability that is at most $1/k^c$, for some positive constant c [10]. The decoder can recover the source symbols from any set of $\Theta = k(1+\varepsilon)$ encoded symbols, with Θ slightly larger than k.

Using the matrix interpretation, the Raptor decoding algorithm corresponds to solving for vector x in the system of equation given in Equation 2 to obtain the input vectors $x_1, ..., x_k$.

C. Validity & Threshold Functions

The validity of both codes of the hybrid FEC code, the Raptor code and the SHA-2 code, have been established. The Raptor codes have been described as a potentially strong and efficient extension of LT codes and it has been proven in [10] that both its encoding and decoding algorithms are linear in time. SHA-2 validity is inferred from its ability to generate a unique value for any set of data with the condition that, given this generated value, it is computationally infeasible to get back the original data [12]. A collision is said to have occurred if two given messages produce the same hash value. To the best of our knowledge, no collision has been reported for SHA-2.

Since the effectiveness of the proposed hybrid FEC code will be constrained by the thresholds of the Raptor code and SHA-2, we need to establish expressions for these thresholds. Any manipulation by jamming beyond what these schemes can tolerate will render the hybrid approach ineffective. The threshold of Raptor code is given by $\Theta = k(1 + \varepsilon)$, which is the minimum amount of the encoded symbols that make the decoding successful. Any manipulation by a jammer that reduces this minimum requirement or delays the arrival of this minimum encoded symbols will adversely affect the performance of the hybrid code. Unlike the Raptor code, a hash function having a message digest of length n-bits has two properties [12]: (i) *One Way*: It will require 2^n evaluations in order to find a message that corresponds to the given message digest. (ii) *Collision Resistant*: Finding two different messages that produce the same message digest, known as a collision, requires an average of $2^{n/2}$ evaluations. A jammer can only manipulate the proposed hybrid code if a collision is found for SHA-2 and also if the jammer is able to satisfy the required number of evaluations to find a message that corresponds to a given hash value.

IV. Simulation & Results

A. The Hybrid Code as an NS-2 Application

We studied the C++ class hierarchy in NS-2 and specified the name of the class for the hybrid code application as "SharapApp" (representing our concatenated code of SHA-2 and Raptor code) and implement it as a child class of "Application". The OTcl (object Tcl) hierarchy name that matches this is "Application/SharapApp". The way that both the sender and receiver behave in the application is implemented in "SharapApp". We first describe the implementation of the main components of the hybrid code application in NS-2 before providing a description of other sub-components.

1) Message Digest Generator: The message digest generator is the SHA-2 component of the hybrid FEC code. This generator is defined in "MessDigest.cc" and "MessDigest.h" of our hybrid FEC code implementation in NS2. The message digest generator is implemented at both sender's and receiver's application layers. A sender's message is received at the application layer and represented as "SharapDATA". The message digest generator is applied on the message, i.e., "SharapDATA", in order to generate the message digest "messDigest" of the sender's message. The hash value in "messDigest" is then stored in the application layer header of the hybrid FEC code implementation. We specify a packet header for the application level communication after the NS2 C++ header structure as "hdr_sharapApp". This application layer header structure is the format used whenever the application has a message to transmit. After storing the message digest of the message using this header structure, the message is handed over to the transport layer agent "UdpSharapAgent" in the "hdr_sharapApp" format.

2) Raptor Encoder: The Raptor encoder is defined in "RaptorEncod.c" and "RaptorEncod.h" of our NS-2 hybrid FEC code implementation. An application data (sender's message + header) sent from the application layer and received at the transport layer of the sender's network by the "UdpSharapAgent" is first broken down into T data blocks. Each block, having a unique ID "BID", is then divided into k input symbols $\Omega = (x_1, .., x_k)$. A symbol is simply a unit of data, meaning that a block comprises of k number of data units. The size of each symbol in bytes is l bytes. Every symbol in a block has unique ID "SID". A symbol is represented in our application as "SharapSymb". To reduce the complexity of fragmenting and re-assembling of "SharapSymb" packets at the sender and receiver's networks respectively, we constrained the size of a symbol to a maximum size $l = 1450$ bytes. This is because the default maximum transmission unit (MTU) of the Ethernet technology is 1500 bytes. We derived the symbol size of 1450 bytes using the formula:

Ethernet MTU - IP_header - UDP_header - Ethernet - Application_header = Symbol_size

which results in 1500 - 20 - 8 - 14 - 8 = 1450 bytes.

The Raptor encoder is made up of two functional modules: (i) pre-code and (ii) LT encoder. The pre-code involves the application of an $n \times k$ binary matrix A, representing the generator matrix of the pre-code on the k source symbols, to obtain intermediate symbols $\rho = (c_1, ..., c_\Theta)$. Matrix A is made up of four essential components. The components are (i) the LDPC generator matrix, (ii) the Gray code generator matrix, (iii) the LT code generator matrix, and (iv) the Robust Soliton Distribution (RSD) [9], which is the degree distribution used for the hybrid FEC code implementation. We used the RSD as our degree distribution because it satisfies the two design requirements of a suitable degree distribution stated in [9]: (i) The degree distribution must ensure that on average few encoding symbols are required to ensure the success of the LT encoding process and subsequently the decoding process. (ii) The degree distribution must ensure that the average degree of the encoding symbols is as low as possible.

The LT encoder stage involves the random selection of the intermediate symbols $\rho = (c_1, ..., c_m)$, with $m < k$,

by the LT encoder to produce the encoded output symbols $\tau = (z_1, ..., z_\Theta)$. A pseudo random number generator is used to generate the random values used in the LT encoder. Each encoded output symbol is generated by the LT encoder randomly selecting a degree d such that d lies between 1 and k from the RSD.

The LT encoder also randomly selects a neighborhood of connected d intermediate symbols. The set of chosen intermediate symbols is then XORed by the encoder to obtain an output symbol z_x. The header of each encoded symbol contains information necessary for a successful decoding of the encoded symbols. The information includes the seed used by the pseudo random number generator "SPRNG", the block identifier "BID", which uniquely identifies the block from which the symbol is obtained from, the "SID" representing the symbol ID, the symbol size l, the degree d of the degree distribution, and the list of neighbor indices "LNI".

3) Raptor Decoder: The Raptor decoder, which operates at the transport layer of the receiver's network, is defined in "RaptorDecod.cc" and "RaptorDecod.h" of the hybrid FEC code implementation in NS-2. The Raptor decoder begins the decoding of the encoded output symbols sent from the sender's network as soon as it receives $\Theta = k(1+\varepsilon)$ encoded symbols. This represents the threshold for the decoding process of a Raptor code. In order for the Raptor decoder to be able to decode the received encoded symbols, it will retrieve the degree distribution d used during the encoding process from the headers of the encoded symbols. It will also retrieve the indices of the set of chosen intermediate symbols, which are also stored in the encoded symbols headers. After retrieving the necessary information, the decoder applies the pre-code on the set of symbols that have been received and then applies the LT code. The decoding process is complete if all the input symbols in a block have been recovered to generate back the original block. Where the decoding process fails, the Raptor encoder waits for more encoded symbols to arrive and tries again the decoding process. If no more additional encoded symbol is received, the CRN can be suspected to be experiencing jamming and the hybrid FEC code recovery block scheme is used to help the system recover from such an attack.

B. Simulation Parameters

Table I presents the different simulation parameters that were used in simulating the hybrid FEC code in the NS-2 extension for CRN.

TABLE I. SIMULATION PARAMETERS FOR THE HYBRID CODE

	Parameter	Value
1	NS2 version used	Version 2.31 for CR
2	Type of Packet	UDP
3	Simulation time	1000 seconds
4	Propagation Model	Shadowing
5	Number of Channels per Radio	2
6	Number of Radios/interface	2
7	Routing Protocol	AODV
8	MAC Protocol	Macng
9	No of CR Nodes	10
10	Simulation Area Size	1000mx1000m
11	UDP Packet Size	1450 bytes
12	Number of Replications	100

In [8] the authors analyzed the different parameters necessary for the successful implementation of Raptor codes, which is also relevant for the hybrid FEC code. The objective of their investigation was to determine the parameters that maximize the speed of data transmission while at the same time decrease the time of decoding the encoded symbols. Essentially, the authors investigated the trade-off between performance, computational complexity and resilience of the Raptor code. These parameters include:

Number of input symbols k in a block: The analysis in [8] asserted that small block sizes are beneficial in realizing high encoding and decoding speed but that block sizes that are too small, i.e., with $k \leq 32$, will cause content switching overhead for the CPU. As a result of this analysis we use block size $k = 1000$ symbols for our simulation.

Length of input symbol l in bytes: In [8] it is stated that the symbol size l has little or no influence on the encoding and decoding speed. But it was observed that when the symbol sizes are significantly large, a slight increase in the speed is noticed. Since the encoding and decoding speed is slightly favored by larger symbol sizes, putting into consideration the Ethernet maximum MTU, we use a length of 1450 bytes, which is the maximum UDP packet size specified for our simulation.

Number of Repair symbols ε: Using the analysis in [8], the number of repair symbols that minimizes the overhead for our chosen block size $k = 1000$ was discovered to be about 16 repair symbols. The analysis shows that 16 repair symbols used for a block size of about 1000 symbols reduces the overhead to about 1.56%. We therefore selected the same number of repair symbols for our simulation.

C. Performance Measurement of the Hybrid Code

We now describe the simulation result of the hybrid FEC code in NS-2 and analyze its performance compared with that of the ordinary NS-2 FEC code with a focus on constant jamming using as performance metrics throughput, packet delivery ration, and packet loss ratio.

1) Throughput: The throughput of a network is defined as the average rate of data successfully delivered by the network. The average throughput is defined as the sum of *packets received* at a layer, e.g., application layer, divided by the *total simulation time*. Figures 2 and 3 present snapshots of the plots of *throughput* against the *simulation time* for the *fair case*, referring to a constant jamming scenario with a jamming rate of 10%. Since we generated a large sequence of the same experiment in our simulation, the cases shown in the figures are the scenarios that are closest to the computed averages of the total scenarios resulting from the same experiment. The case of the ordinary FEC code is depicted in Figure 2, where the code fails at every instance of transmissive value fault. This is because any decoded data made up of corrupted encoded symbols is dropped by the CRN. The ordinary FEC code also fails whenever there is a decoding failure, i.e., when the number of encoded symbols recovered at the destination is less than k.

The impact of the same jamming scenario using the hybrid FEC code is shown in Figure 3. One can see that the hybrid FEC code was able to mitigate most jamming caused by the constant jammer. The instances where the hybrid FEC code

fails, represented by zero throughput in the figure, were results of one or combinations of the following five factors resulting from the implementation of the recovery block for the hybrid FEC code:

• The minimum encoded symbols $k(1+\varepsilon)$ received for a successful decoding by the hybrid FEC code were corrupted or manipulated (value fault). Therefore, the hybrid code waited for more encoded symbols, i.e., increasing the value of ε, thereby introducing a type of delay we refer to as "Iterative Decoding Delay" (IDD). This accounts for the failure of the network to deliver packets to the receiver whenever such delay is experienced by the CR Network.

• Delay resulting from the time taken by CR nodes to relocate from a suspected or potential source of jamming.

• Delay resulting from the time used by the CRN to re-authenticate participating CR nodes in order to isolate a potential jammer.

• Delay resulting from the time to switch from a jammed CCC to another channel free of jamming by CR nodes.

• The failure of the recovery block due to non-implementation of the retransmission procedure when UDP data is corrupted. This retransmission procedure will only be implemented in an application that is not delay sensitive.

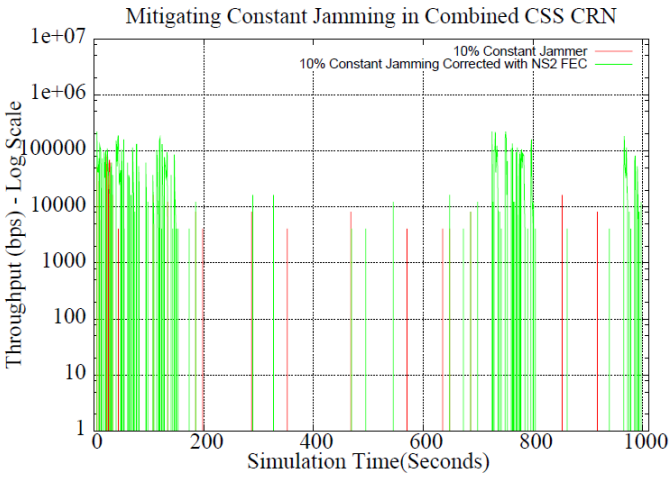

Fig. 2. NS2 FEC Code Performance against 10% Constant Jamming

Figures 4 and 5 depict the snapshots for the *moderate case* of constant jamming. This is when the rate of jamming was 30%, resulting in increasing instances of value faults. In Figure 4 it can be seen that the throughput of the ordinary FEC code dropped significantly as the code fails at every instance of transmissive value fault and whenever there is a decoding failure of the FEC code. The hybrid FEC code significantly outperforms the ordinary FEC code as seen in Figure 5, where it was able to deliver higher throughput during constant jamming. Nonetheless, the hybrid FEC code fails in intervals, seen with zero throughput, due to the actions of the aforementioned recovery block.

Figure 6 shows throughput during the *worst case* scenario, where constant jamming is 100%. This is a total jamming scenario that resulted into a zero throughput of the CRN

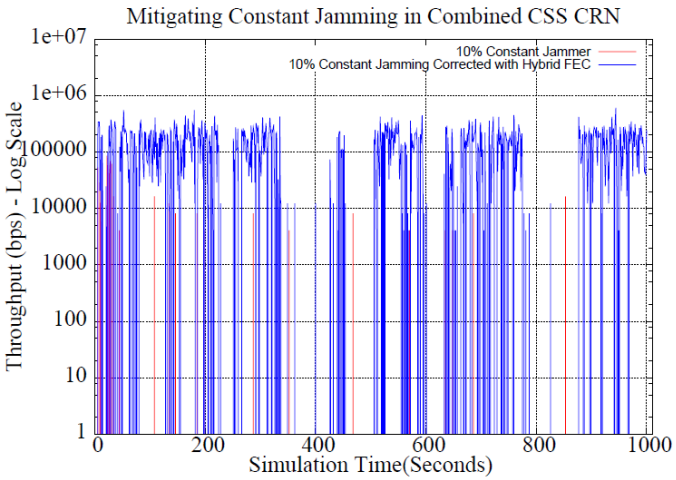

Fig. 3. Hybrid FEC Code Performance against 10% Constant Jamming

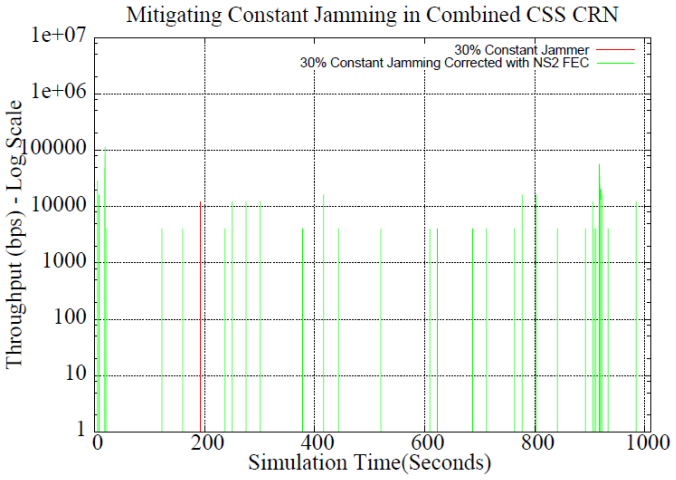

Fig. 4. NS2 FEC Code Performance against 30% Constant Jamming

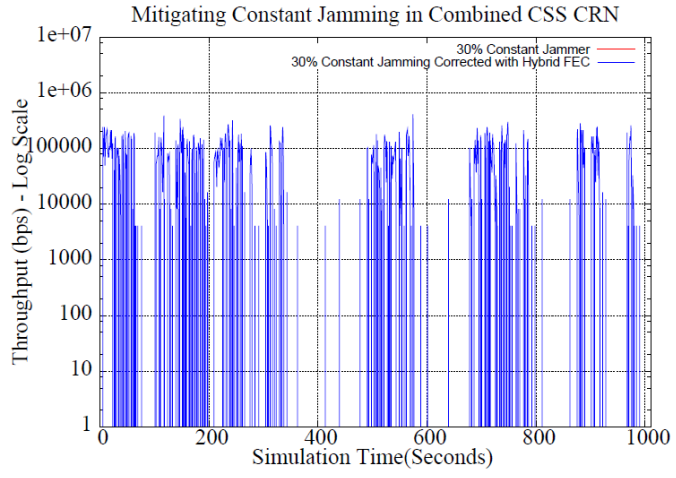

Fig. 5. Hybrid FEC Code Performance against 30% Constant Jamming

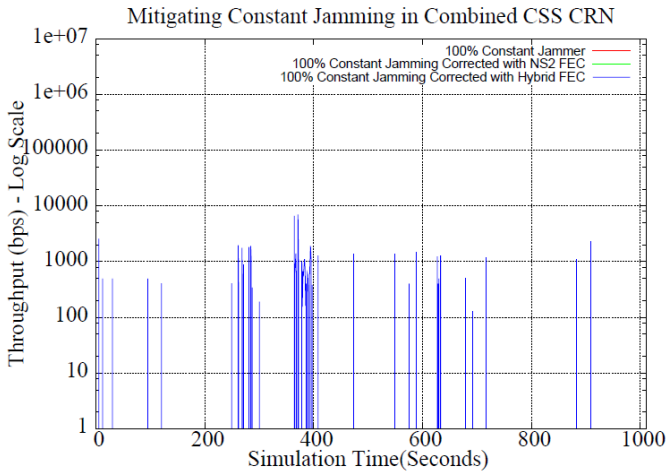

Fig. 6. Hybrid FEC Code Performance against 100% Constant Jamming

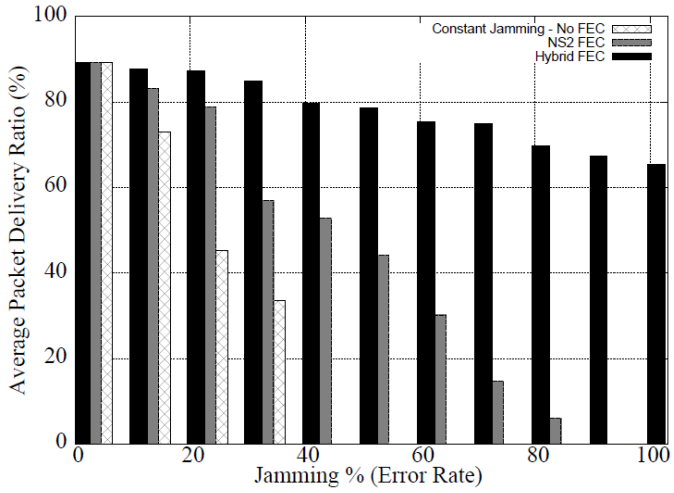

Fig. 7. PDR measurement of Hybrid FEC Code Performance against Constant Jamming

without the implementation of the hybrid FEC code. The figure shows that the ordinary FEC code was not able to mitigate any instance of the 100% constant jamming and as a result, no packet was delivered at the receivers by the CRN. In the case of our hybrid FEC code, small spurts of packets were delivered during the simulation, but fails intermittently as it tried the recovery block procedures to cope with the high rate of data corruption or manipulation of the constant jammer.

In summary, we observed that the hybrid FEC code largely out-performs the ordinary FEC code as it successfully handles most incidences of omissive and transmissive faults. It is also noticeable that even when the rate of jamming was 100% characterized by increased rate of data corruption and manipulation, the hybrid FEC code was still able to recover some amount of data packets during the jamming, as they were successfully delivered to the destinations. The hybrid FEC code out-performs the NS-2 FEC because of the following reasons: the efficient design of the hybrid FEC code algorithm together with the recovery block implementation, and the careful selection of simulation parameters that lead to efficient performance of the hybrid FEC code.

2) Packet Delivery Ratio (PDR): The PDR is defined as the ratio of the sum of *packets received* at a layer to the sum of *packets sent* in the same layer. The PDR distinguishes between a congested network and a network under jamming attack, as a highly congested network can still produce a high PDR. The *recovery rate* of an FEC code is the ratio of the number of received encoded symbols to the total number of source symbols. A low recovery rate arises if the number of received encoded symbols is less than the number of source symbols. Therefore, the efficiency of a FEC code is characterized by its recovery rate.

Figure 7 shows the average PDR for the total number of replications for different jamming rates, comparing the ordinary FEC code with the hybrid FEC code. The superior performance of the hybrid FEC code over the ordinary FEC code is immediately noticeable. The PDR of the ordinary FEC code drops significantly as the jamming rate increases. At 50% jamming, the PDR of the ordinary FEC is about 45%. At the jamming rate of 100%, the PDR of ordinary FEC drops to zero, meaning that the ordinary FEC code breaks down completely. Unlike the ordinary FEC, the hybrid FEC, which maintains very high PDR of about 70% even when the jamming rate increases to 90%. It is also noticeable from the figure that at a worst case jamming rate of 100%, i.e., total jamming, the hybrid FEC code still maintains a high PDR of about 65%. These high PDRs recorded by the hybrid FEC code mean that the hybrid code is robust against DoS activities of constant jammers as the hybrid FEC code was able to recover and deliver a high percentage of data that was successfully sent by the sender. When looking at the 100% jamming scenario the reader should not get confused by the high PDR and the sparce throughput in Figure 6, as the network can have a low throughput but a high PDR.

3) Packet Loss Ratio (PLR): The PLR is defined as *Number of lost packet / (Number of lost packet + Number of packets received successfully)*. The PLR is closely related with the quality of service (QoS) of the hybrid FEC code. It also gives a more accurate estimation of the recovery rate of the hybrid code because it does not put into consideration the number of data packets sent from the sender, but calculates the rate at which the hybrid code is able to recover successfully encoded input symbols at the receiver. A high recovery rate implies that encoding, decoding, SHA-2 hash computation and the processes through the action of the recovery block for the hybrid FEC code are computationally efficient in the sense that most of the encoded input symbols were successfully recovered without error at the receiver. A low PLR indicates that a high percentage of the packets were received either uncorrupted or that the corrupted packets were recovered by the recovery action of the hybrid code's recovery block. The relationship between the recovery rate of the hybrid FEC code and the PLR can thus be stated as *Recovery Rate = 1 - PLR*.

Figure 8 is the plot of *average packet loss ratio (PLR)* against *jamming percentage*. The average represents the average PLR for the total number of simulations for the same experiment for each rate of jamming. It can be seen in the figure that the hybrid FEC code maintains a very low PLR for the different jamming rate of the constant jammers. The

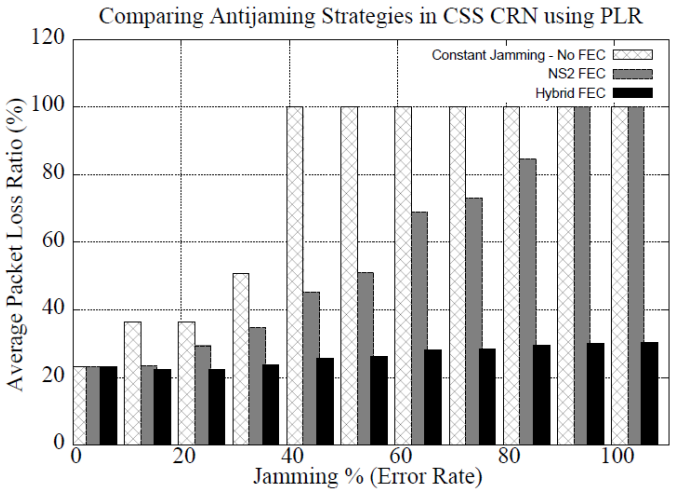

Fig. 8. PLR measurement of Hybrid FEC Code Performance against Constant Jamming

ordinary FEC code has low PLR up to a jamming rate of 30%, but beyond this the code breaks down significantly as the PLR increases to close to 70%, even when the jamming rate is just about 60%. Since low PLR implies a high recovery rate, it follows that the hybrid FEC code maintains high recovery rate even when the rate of jamming is 100%. The high recovery rate of the hybrid FEC code recorded against that of the ordinary FEC code implies that the algorithms of the encoding, decoding and SHA-2 hash computation processes through the action of the recovery block for hybrid FEC code are computationally more efficient and robust against the different rate of jamming than that of the ordinary FEC code.

4) Summary of Statistical Analysis: Table II presents the summary of the statistical analysis of the result of our simulations for constant jamming with respect to PLR. Since PLR gives a good estimate of the recovery rate of the hybrid FEC code, and hence the efficiency of its encoding and decoding algorithms, we only present the PLR analysis for constant jamming. The 10% jamming represents the fair case of constant jamming, the 30% jamming represents the moderate case of constant jamming and the 100% represents the worst case of constant jamming.

TABLE II. SUMMARY OF STATISTICAL ANALYSIS FOR AVERAGE PLR - CONSTANT JAMMING

	Min.	Max.	Ave.	Stand. Dev.
0% Jamming	20.73	25.87	23.24	1.19
10% No FEC	32.15	48.84	36.28	1.67
10% NS FEC	22.02	30.76	23.56	1.22
10% Hybrid FEC	21.18	26.42	22.26	1.40
30% No FEC	50.00	62.73	50.70	1.56
30% NS FEC	32.03	37.53	34.86	1.64
30% Hybrid FEC	22.49	27.86	23.83	1.51
100% No FEC	100	100	100	0
100% NS FEC	100	100	100	0
100% Hybrid FEC	28.54	33.43	30.42	1.61

V. CONCLUSIONS

In this paper, we investigated the performance of CSS CRN operating in the presence of CR constant jammers that are capable of introducing transmissive and omissive value faults. We discovered that existing anti-jamming solutions are not able to handle these fault types and consequently proposed a hybrid FEC code capable of mitigating the entire class of faults identified in this adversarial model. We simulated and investigated the performance of the proposed solution in NS-2. The result of our analysis shows that our proposed solution is efficient and robust against any level of DoS caused by constant jammers in a CSS CR network. We shall be presenting the result of our simulation and analysis for other jamming types in separate papers.

REFERENCES

[1] FCC Spectrum Policy Task Force, *Report of the spectrum efficiency working group*, ET Docket no. 02-135, Nov. 2002. [Online]. Available: http://www.fcc.gov/sptf/reports.html

[2] I. F. Akyildiz, B. F. Lo, and R. Balakrishnan, *Cooperative Spectrum Sensing in Cognitive Radio Networks: A Survey*, Physical Communication (Elsevier) Journal, vol. 4, no. 1, pp. 40-62, March 2011.

[3] V. Balogun and A. Krings, *On The Impact of Jamming Attacks on Cooperative Spectrum Sensing in Cognitive Radio Networks*, Proc. 8th Annual Cyber Security and Information Intelligence Research Workshop, Oak Ridge National Laboratory, January 8 - 10, 2013.

[4] V. Balogun and A. Krings, *An Empirical Measurement of Jamming Attacks in CSS Cognitive Radio Networks*, Submitted to the 27th IEEE Annual Canadian Conference on Electrical and Computer Engineering (CCECE 2014), Toronto, Canada, May 5 - 8, 2014.

[5] J. L. Burbank, et al., *A Common Lexicon and Design Issues Surrounding Cognitive Radio Networks Operating in the Presence of Jamming*, IEEE MILCOM 2008; 1-7.

[6] W. Cadeau and X. Li, *Anti-jamming Performance of Cognitive Radio Networks under Multiple Uncoordinated Jammers in Fading Environment*, Proc. of the 46th Annual CISS, Princeton Univ., NJ, March 2012.

[7] Qi Dong and Donggang Liu *Adaptive Jamming-Resistant Broadcast Systems with Partial Channel Sharing*, Proc. 2010 International Conference on Distributed Computing Systems, Genova, Italy.

[8] Philipp M. Eittenberger, Todor Mladenov, and Todor Mladenov, *Raptor Codes for P2P Streaming*, 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2012

[9] M. Luby, *LT-codes*, in Proceedings of the 43rd Annual IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 271-282, 2002.

[10] Shokrollahi, A. *LDPC Codes: An introduction*. Available: www.ipm.ac.ir/IPM/homepage/Amin2.pdf.

[11] A. Shokrollahi, *Raptor codes*, IEEE/ACM Transaction on Networking, 14(SI):2551-2567, 2006.

[12] N. Sklavos and O. Koufopavlou, *Implementation of the SHA-2 Hash Family Standard Using FPGAs*, J. of Supercomputing, Vol. 31, No 3, pp. 227-248, 2005.

[13] National Institute of Standards and Technology (NIST) *Secure Hash Standard (SHS)*, Federal Information Processing Standards Publication, March 2012. Available: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[14] W. Zhang and K. Letaief, *Cooperative Spectrum Sensing with Transmit and Relay Diversity in Cognitive Radio Networks*, IEEE Transactions on Wireless Communications 7 (12) (2008) 4761-4766.

[15] R. W. Doran, *The Gray Code*, CDMTCS Research Reports, CDMTCS-304, March 2007.
Available: http://www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports/304bob.pdf

[16] A. Ghasemi and E. Sousa, *Asymptotic Performance of Collaborative Spectrum Sensing Under Correlated log-normal Shadowing*, IEEE Communications Letters, Vol. 11, No. 1, pp. 34-36, 2007.

[17] B. Randell and J. Xu, *Recovery Blocks*. In: Marciniak, J.J, ed. Encyclopedia of Software Engineering. New York, USA: Wiley, 1994, pp.1037-1038.