Concurrent Code Spread Spectrum:

Theory and Performance Analysis of

Jam Resistant Communication Without Shared Secrets

by

WILLIAM LOUIS BAHN

B.Sc., Colorado School of Mines, 1991

M.Eng., Colorado School of Mines, 2004

A dissertation submitted to the Graduate Faculty of the

University of Colorado at Colorado Springs

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

2012

This dissertation for Doctor of Philosophy degree by

William Louis Bahn

has been approved by the

Department of Electrical and Computer Engineering

by

Mark A. Wickert, Chair

Rodger E. Ziemer

Charlie Wang

C. Edward Chow

Xiaobo Zhou

Leemon C. Baird III

Date

Bahn, William Louis (Ph.D., Electrical Engineering)

Concurrent Code Spread Spectrum:
Theory and Performance Analysis of
Jam Resistant Communication Without Shared Secrets

Dissertation directed by Professor Mark A. Wickert

As commercial and military reliance on wireless communications grows, jam resistance becomes increasingly important. Jam resistance in omnidirectional radio communications relies on spread spectrum techniques that employ symmetric keys (i.e., shared secrets). Key management is a challenging problem in scaling up critical networks such as the Global Information Grid (GIG). While alternatives to symmetric keys utilizing asymmetric cryptography and a Public Key Infrastructure (PKI) exist for the Traffic Encryption Keys (TEK) used to encrypt, decrypt, and authenticate transmissions, the same cannot be said for the Transmission Security Keys (TSK) that protect the waveform from hostile jamming.

An unstated, but widespread, assumption exists that spread spectrum systems require symmetric keys to achieve jam resistance. While such shared-secret schemes are workable in small networks, the scale and nature of theater-wide, mobile, ad hoc wireless networks will quickly overwhelm any practical key management strategy. The problem could be alleviated if an asymmetric system for the physical layer were developed, but little attention has been directed at this problem and no such solution exists. Making matters worse is that public-access systems - such as the civilian side of the Global Positioning System (GPS) - preclude reliance on secret keys since every person is an authorized user. Yet while these systems are recognized as having little to no jam resistance, they are becoming increasingly critical to activities such as civilian aviation operations.

Presented here is a new form of spread spectrum based on a new coding theory - the theory of concurrent codes - that permits the construction of jam-resistant physical layers that not only do not rely on shared secret keys, but rely on no keys at all. One particular unkeyed concurrent algorithm, the BBC algorithm, is explored in depth. Then the performance of BBC pulse-based concurrent code spread spectrum (CCSS) is also analyzed for the case of additive white Gaussian noise (AWGN) barrage jamming and compared to similar frequency hop (FHSS) and direct sequence (DSSS) systems. Finally, prototype systems that demonstrate the capabilities of concurrent codecs using audio, image, and radio frequency transmission channels will be discussed.

DEDICATION

This work is dedicated, first and foremost, to the memory of my father, Donald K. Bahn, for his steadfast devotion to his family through years of challenging times that most men would not have endured and the solid core of integrity that was the center of his being. I can't begin to imagine how different my journey through life might have been but not for his every-day consistent and reliable example.

I further dedicate this to my wife, Phoebe, without whose encouragement this might never have reached a conclusion.

Finally, I wish to express my extreme appreciation for the tolerance and understanding of my advisor, Mark Wickert; my entire degree program would have ended abruptly on multiple occasions as I fought health and other concerns but for his willingness to continue working with me on each renewed attempt with a level of patience that was as remarkable as it was undeserved.

ACKNOWLEDGEMENTS

The author acknowledges the critical role in this work of the Principal Investigator, Dr. Leemon C. Baird III and also the contributions of the third member of Team BBC, Mr. Michael D. Collins.

This work was sponsored in part by the 688th Information Operations Wing (688IOW), Lackland AFB, TX, and was performed principally at the Academy Center for Cyberspace Research (ACCR) at the United States Air Force Academy.

Contents

List of Figures

List of Tables

Chapter 1

Preface

Concurrent Coding Theory is a new coding theory developed at the United States Air Force Academy's (USAFA) Academy Center for Cyberspace Research (ACCR). Because it is relatively unknown, its practical aspects will be covered in some depth. Since the author was one of the primary researchers involved in its development this is more than mere background material; however, the focus of the dissertation is on the realization and preliminary characterization of concurrent code spread spectrum (CCSS). This is a new form of spread spectrum based on concurrent codes that, unlike traditional forms, does not rely on shared secrets for its jam resistance. As a result, the scalability limitations imposed by symmetric key distribution are almost entirely eliminated.

This work is partitioned as follows:

The **Introduction** provides an overview of how radio communications are increasingly being used and why that translates into a corresponding increase in the importance of incorporating jam resistance into many of these systems.

The **Overview of Jam Resistance** discusses the concept of jam resistances within the context of its goals and how it fits within the overall framework of electronic counter measures and counter-counter measures. Included is a description of present

techniques used to achieve jam-resistance and their shortcomings, both intrinsic and in practical terms in light of current trends. The emphasis is on omnidirectional wireless communications, as these are the types of communications that present capabilities are least able to protect. The primary limitation, relevant to this work, is the reliance of shared secrets for jam resistance due to the poor scalability of symmetric keys.

The **Conceptual Framework** for jam resistance without shared secrets is drawn using a useful parallel to how the symmetric key distribution problem has been largely overcome in the information security arena. This sets a framework for viewing how this research permits the mitigation of this same problem in the physical waveform construction. With this in mind, the requirements of an unkeyed jam-resistant channel are established by examining the operating environment that such channels will face. This serves as a guide to determine what other relevant work exists that could offer insights into this problem and potential solutions.

The **Literature Search** discusses the body of work that presently exists relating to superimposed codes, of which concurrent codes are a new subset. This makes clear that concurrent codes, and thus anything based on them, represent an extension to the body of knowledge.

The **Concurrent Coding Theory** introduces the underlying concepts related to concurrent codes. Because these concepts require, for most people, subtle shifts in thinking about encoding and recovering information, they are built incrementally using toy examples.

The **BBC-based Codecs** are explored in depth. First, the Standard BBC encoding and decoding algorithms are presented and characterized in depth. Then several extensions to the standard algorithm are discussed as are several task-specific processing blocks that can enhance the performance of BBC-based codecs.

The **RF Implementations** look at how the rather unconventional requirements of concurrent-code based communications can be realized in practical radio systems.

After discussing several potential implementations at a high level of abstraction, a detailed look is taken at one particular implementation, namely pulsed-BBC, which is analyzed in some depth.

The **Performance of Pulsed-BBC CCSS** in AWGN barrage jamming first analyzes the equivalent bit error rate (BER) performance of Pulsed-BBC CCSS, including taking into account inherent differences between concurrent channels and traditional modulation schemes. Then optimal jamming signals for this system are discussed, followed by a comparison of the performance of Pulsed-BBC CCSS to traditional frequency hop spread spectrum (FHSS) and direct sequence spread spectrum (DSSS) systems.

The **Demonstrators** that have been developed to demonstrate key aspects of concurrent codes and, specifically, the BBC algorithms are discussed. These include a visualization applet and a sonic demonstrator, as well as actual RF demonstrators using software-defined radios.

The **Conclusion** makes some general closing remarks as well as discussing potential avenues for continued research.

Chapter 2

Introduction

Reliable communication of data - be it voice, imagery, telemetry, financial, command and control, or an almost countless variety of other types - has become critical to modern economies and also to modern warfare [1, 2]. Long gone are the days when interfering with an opponent's radio communications seldom proved more than an annoyance or caused more than relatively minor delays in the flow of information. Today we are faced with many examples of communication systems in which even very short term interruptions can prove catastrophic. For instance, an airliner low on fuel in bad weather would be in a very precarious situation if the navigation signals were interfered with during final approach. Similarly, a battlefield engagement can go horribly awry if the communications used to distinguish friend from foe are compromised. These are just two examples where malicious jamming of a communications resource at a critical time can result in the loss of life.

To further expand on these two examples, the 1996 Federal Radionavigation Plan [3] specifically called for the transition to the Global Positioning Satellite (GPS) system as the primary means for enroute and terminal navigation for civil aircraft operations within the National Airspace System (NAS) with the phaseout of tradi-

tional VOR[1]/DME[2], NDB[3], and ILS[4] services to begin in 2005 and be complete in 2010. Further, it called for the use of GPS as a "sole-means" navigation service for all phases of flight, meaning that aircraft would be permitted to operate using GPS navigation without any requirement for a backup means. Subsequent plans reveal significant reflection on the wisdom of using GPS as a sole-means system in light of the recognized vulnerabilities in the system, including the lack of jam resistance. Despite this, the present 2010 Federal Radionavigation Plan [4] still calls for significant reductions in the installed base of other systems as use shifts to GPS and a more gradual phaseout of many of these systems is slated to begin in 2011 or shortly thereafter. The expectation is for these systems to be progressively designated as back-up facilities and taken off-line through attrition. As this happens, GPS jamming will become an increasingly attractive option for terrorists.

On the battlefield, the United States is presently pursuing the doctrine of net-centric operations (NCO) (a.k.a., net-centric warfare) with the goal of realizing the Global Information Grid (GIG) [5, 6] that will encompass users at all levels including top-level commanders, soldiers in the field, and conceivably even individual pieces of equipment such as rifles and artillery shells. Parts of the GIG have already been implemented and proven to be of significant utility. As the GIG is further realized, significant gains in efficiencies can potentially be achieved. For instance, if (among many other pieces of data) every rifle is reporting its ammunition usage in realtime via the GIG, then the flow of battle can more effectively be monitored and managed by everyone from squad leaders to theater commanders. In addition, logistics personnel can more rapidly predict the need for specific supplies and more quickly preposition them close to where they will be needed. The result is a "force multiplier" in which

[1]VHF Omnidirectional Range - Allows aircraft to determine their absolute direction from the station.

[2]Distance Measuring Equipment - Allows aircraft to determine their distance from a VOR station.

[3]Non-directional Beacon - Allows aircraft to determine their relative direction from the station.

[4]Instrument Landing System - Presently the primary means for performing precision instrument landing approaches to an airport.

fewer people with less equipment and fewer supplies will be able to accomplish the same missions as before. However, it also creates a critical reliance on the GIG where any significant disruption can threaten mission accomplishment.

Clearly, any link that forms a critical path in any mission-critical communication system will need to be designed and employed with jam resistance in mind. This is particularly true in adversarial environments, such as anti-terrorism or tactical military operations, where malicious players exist whose objectives would be furthered by the disruption of those links.

Chapter 3

Overview of Jam Resistance

To appreciate what "jam resistant" means, it is necessary to adequately establish what "jamming" means. Within the framework of the four classic goals of secure communications (availability, confidentiality, authenticity, and integrity), jamming is an attack on the availability of a communications resource. In the parlance of the communications community, jamming is an example of an electronic counter measure (ECM) since its intent is to counter the goals of the enemy's communications by electronic means. Electronic efforts to mitigate jamming fall into the category of electronic counter counter measures (ECCM). Of course, such measures invite the development of more effective measures to counter those, which could be called ECCCM, and so on in an endless progression. The point is that neither side operates in a vacuum and the abilities and limitations, electronic and otherwise, of the opposing side must be taken into consideration, at least in general terms.

Jamming generally involves artificially injecting a second signal into a communications channel that combines with the legitimate signal in such a way that the receiver's ability to recover the correct information is significantly impaired. This second signal can be as simple and brute force as wideband noise transmitted with sufficient power

to overwhelm the legitimate signal, or it might consist of very carefully crafted signals intended to mislead the receiver into misinterpreting the combined signal it receives.

Brute force jamming with broadband noise is arguably the surest way to jam a radio broadcast. It also tends to be one of the most costly by several measures. The raw energy requirements alone are one potential issue, especially if the jamming platform is small, remotely deployed, and/or battery-powered. Furthermore, the power required for effective jamming may be sufficient to permit the detection and localization of the jamming source. In most civilian environments this increases the risk of apprehension and prosecution for those responsible; on the battlefield, consequences tend to be more direct and permanent.

Attackers therefore prefer jamming methods that minimize energy expenditure, if for no other reason than self-preservation. The ideal method would permit a minuscule amount of energy, significantly less than that used by the legitimate sender, transmitted in a single burst to corrupt a legitimate message just enough to prevent its accurate reception. Even if the receiver identifies a message as being corrupted beyond recovery, they must still request a retransmission (which may also be initiated by the sender if an expected acknowledgement fails to arrive). This reduces the effective bandwidth, and hence availability, of the channel.

It is important to avoid equating jam resistant with jam proof. If the attacker is willing and able to commit sufficient energy and resources, they will always be able to sufficiently interfere with a legitimate signal so as to render it useless - this is true even for hardwired networks. As in many adversarial environments, the goal in jam resistance is not to create a situation in which the opponent absolutely cannot triumph, but rather one in which the cost of doing so is prohibitive.

The cost of jamming is a complex and subjective computation involving such factors as time, effort, resources, energy, and risk. While the defender, in theory, only has to make the cost of one component sufficiently high, the attacker can respond by

trading reliance on that component for others. Again, the classic game of counter-measures, counter-counter-measures, and so on.

With this understanding comes the realization that characterizing the jam resistance of any communications channel is neither trivial nor static. In general, it must take into account the type of attacks that are possible, the various costs associated with them, and the type of counter-measures that are available. Most of that is well beyond the scope of this work and hence the measure of jam resistance that will be used will necessarily be much more narrow and abstract.

A reasonable measure of jam resistance relates the effort required by the receiver to recover the original message to the effort expended by the attacker. As previously noted, however, "effort" is a very elusive term in this context. For our purposes, we will use two complementary measures for "effort required by the receiver." The first of these is the effective bit error rate of the channel, which is the measure commonly used. The second is the computational effort required to recover the legitimate message, which is generally ignored but which, for reasons explained later, cannot be ignored here. Turning to the "effort expended by the attacker," this will be measured using the conventional metric of the ratio of the mean power of the attacking signal to that of the legitimate signal at the receiving antenna.

There are three basic techniques that can be used to provide resistance to hostile jamming: power, directionality, and jam-resistant waveforms. All three are complements and can be combined, to the degree feasible, to achieve the maximum level of jam resistance possible.

3.1 Power to Achieve Jam Resistance

The effectiveness of a jamming signal is directly related to the ratio of the power in the jamming signal to the power in the legitimate signal. Thus, "simply" turning

up the transmit power provides resistance to hostile jamming. However, there are both physical and practical limits to this approach. Any given transmitter will have a maximum power output that it can produce while still maintaining the necessary waveform quality. Furthermore, high transmission power may be contraindicated when a goal is for the transmissions to remain undetected or to minimize the chance of attracting unwanted attention (such as anti-radiation missiles) from the adversary.

3.2 Directionality to Achieve Jam Resistance

A high degree of jam resistance can be attained by the use of highly directional signal propagating elements - if the attacker can only interfere with the signal from a limited number locations, then the defenders gain a significant advantage by controlling where those locations are and ensuring that the attackers do not have easy access to them. For instance, satellite downlinks can be made quite jam resistant by using high-gain receiving antennas. By having a very narrow reception arc that is directed well above the horizon, the attacker's problem of injecting a malicious signal into that arc is a challenging one, particularly if the receiver is airborne. While not as effective, purely terrestrial channels can be made quite jam resistant by employing very narrow beams of energy, such as radio-frequency or laser light. This also includes the growing range of beam-forming and steering technologies, both for transmission and reception. Another very effective means is the use of wired connections, such as copper or fiber optic, to carry the traffic. While not typically thought of in those terms, it is difficult to imagine anything more highly-directional than a piece of wire. Similarly, a physical message courier can be categorized as highly-directional.

It's worth noting that channels that base their jam resistance on high directionality are really relying on preventing the attacker from injecting the jamming signal at all

- they are not necessarily any more successful than channels that claim no particular jam resistance at tolerating those jamming signals that do get injected.

While highly-directional links offer real benefits, they are not without serious shortcomings as well. In particular, they are typically not very scalable or flexible. A significant amount of preparation is usually involved establishing them, the number of users they can accommodate is relatively limited, and they generally cannot be readily adapted to highly dynamic environments.

3.3 Shared Secrets to Achieve Jam Resistance

Reliance on using high transmit power and/or directionality for jam resistance is not always an option. In particular, jam resistance must extend to relatively low-power omnidirectional radio links as well because their scalability, simplicity, and flexibility will virtually assure their use, possibly even dominance, in the types of large-scale, flexible, highly-mobile battlefield ad hoc networks that will be found in tactical environments.

The strict definition of "omnidirectional" requires that signal propagation (transmitter) and sensitivity (receiver) patterns be independent of direction. In practice, true omnidirectionality is nearly impossible to achieve. This is particularly true in radio-frequency systems, where a system is generally considered omnidirectional if it has little directional dependence within a given plane (generally horizontal). Omnidirectional systems are, by nature, wireless. The reverse, of course, is not true; wireless communication channels can most certainly be highly directional. For this treatment, a system will be considered omnidirectional if the intended recipients have no preferential access to the transmitter's signal relative to unintended recipients and, similarly, if the receivers are as sensitive to an attacker's signal as they are to that coming from the legitimate sender. These are reasonable assumptions in many tac-

tical environments where the rapid, unpredictable movements of ally and adversary alike frequently make this a practical reality.

Omnidirectional alternatives generally bring extensive simplicity and flexibility to the situation, but suffer because the attacker has essentially the same access to the channel as the legitimate parties and, hence, can detect what the sender is transmitting and affect what the recipient is receiving fairly easily. To overcome this disadvantage, legitimate parties can collude ahead of time and share special knowledge, known as a "shared secret" or "symmetric key", about how the channel will be used. This information is used to effectively hide the true signal within a much broader piece of spectrum. The broader the bandwidth over which the signal is spread, the greater the level of jam resistance since an attacker must also spread their signal over a similar bandwidth. If they are successful at depriving the attacker of this knowledge, they can shift the odds significantly in their favor. For instance, in the case of spread spectrum the hop sequence, spreading code, or other information used to spread the spectrum is assumed to be such a secret and virtually all jam resistance disappears if that secret is compromised.

This is a reasonable point to make the distinction between the different kinds of keys typically used in a secure radio network. For our purposes, these can be divided into two groups: TSKs and TEKs. A transmission security key (TSK) is used to protect the physical layer signal of the communication system, while a traffic encryption key (TEK) is used to protect the information content conveyed by that signal. Except where specifically noted, this paper focuses exclusively on the TSKs. Current TSKs are "symmetric keys" that must be closely held and safeguarded, on the one hand, while being widely and efficiently distributed, on the other. These are conflicting goals that, as the pool of authorized users grows, drive design options in opposite directions: satisfying the increased security demands slows the distribution

process while larger networks generally require more agile and rapid distribution of key material.

The challenges of managing symmetric keys on even modest scales is a well-known problem in the Information Assurance community and present solutions do not scale well to large pools of authorized users [7, 8]. Yet tactical wireless networks are growing increasingly large, particularly if they include distributed sensors. They also involve increasingly diverse participants, such as members from other agencies or coalition partners. The ability to maintain a sufficiently agile key distribution system while simultaneously making it sufficiently secure is far from guaranteed. Furthermore, there are public access systems, such as civilian-GPS and cellular networks, for which the use of shared secrets is impossible or extremely impractical; these systems could now enjoy the benefits of jam resistance.

3.4 Key Management Problem

While effective, the use of symmetric keys also serves as an Achilles' heal. Should the attacker gain access to the key, they gain the ability to jam the channel. A large part of the defender's task therefore becomes key management - distributing keys to legitimate users, denying them to unauthorized parties, detecting when keys have been compromised, and ensuring that legitimate users do not use compromised keys.

There are several obstacles to achieving proper key management, the most significant of which are the distribution, security, and perishability problems. The logistics involved in simply distributing the keys becomes a major problem as the size of the network grows. It is one thing to give all the members of a special operations team or a flight of fighter aircraft the keys they will need to communicate amongst themselves just prior to a mission; it is quite another to ensure that the front-line soldiers from an allied nation have the keys necessary to access real-time target intelligence coming

from yet another allied nation half a world away. Another major problem is keeping the keys secret - the more people that have access to the key, and the longer ahead of time they have that access, the greater the likelihood that the key will be compromised. To address this, it is necessary to have strict accountability for the keys. In small scale networks this is reasonably doable, but the problem quickly becomes untenable as the number of people with access grows. Related to this is the problem of detecting when keys have been compromised - perhaps the only thing worse than the enemy learning your secrets is for the enemy to do so while you still believe they are secure. While small teams of people are much more likely to know when their keys have been compromised (compared to large networks), it is still generally the case that sensitive keys are only used for a short period of time - in essence it is assumed that the adversary will obtain the keys after a certain period of time and hence they are thought of as perishable commodities. When all three of these problems are combined, it becomes self evident that the need to rapidly distribute short-lived keys to large numbers of people in a secure fashion is not only challenging, but a practical impossibility for all but the smallest of networks.

In some instances key management is a moot point because reliance on shared secrets is simply incompatible with certain types of communications - in particular, public-access systems such as civilian-side GPS have authorized user pools that include every person on the planet. To the degree that those systems are also omnidirectional, they have essentially no jam resistance at all. Since incorporating a high degree of directionality will not always be practical (or even possible) the need for jam-resistant omnidirectional communications between parties having no prior shared secret will only grow.

Chapter 4

Conceptual Framework

4.1 Typical Spread Spectrum System

A simplified block diagram of a typical spread spectrum system relevant to this dissertation is shown in Figure 4.1. The term "*codec*", which is short for "coder/decoder", will be used throughout this dissertation in a very general sense, namely a device that transforms data from one digital format to another. Some of the codecs in the diagram are more commonly known by other names (which will be mentioned), but we want to emphasize that they are all performing the same basic function and that the traditional partitioning is therefore somewhat arbitrary and flexible. The reason for this approach is that we use a slightly repartitioned framework. In the following discussion, we will be referring to the transmit path when terms such as 'input' and 'output' are used. Each element in the receive path, of course, is simply undoing what the corresponding transmit element did.

The "*data source*" can be anything from a microphone to a camera to a hard disk, as long as the data is in digital form. It is commonly the case that such data is in a format that, while convenient for performing processing related to the type of data, for instance audio or video editing, is not very efficient in terms of the number of bits

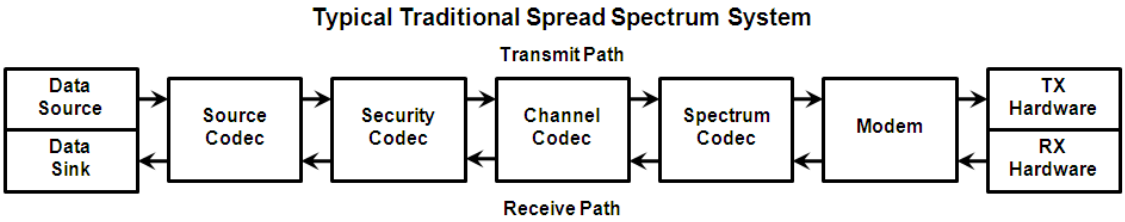

Figure 4.1: Simplified spread spectrum system.

used to represent the information contained. From an information theoretic point of view, each "*symbol*" in the data stream has a relatively low "*entropy*", which is a measure of the average uncertainty, and hence information content, of each symbol [9]. For instance, consider an audio recording stored in a raw format where each sample is represented as a 16-bit integer. The values of two successive samples are unlikely to differ by a large amount since both are nearby samples from a continuous waveform. Thus, it is not surprising that given one sample, we can predict with a high degree of certainty what many of the bits in the next sample are likely to be. Thus, those bits contain very little new information and, in the sense that they largely tell us what we already know, are redundant.

Let's consider a simple example to see how each block in the transmit path affects it. Our data source will be some piece of English text represented using ASCII, which encodes each character using an 8-bit byte. Coming out of the data source let's say that there are eight thousand bits of data (one thousand characters) that we wish to transmit from Point A to Point B. The "*source codec*" is designed to remove as much redundancy in the source data as possible so as to minimize the number of bits of data that must be transmitted. Ideally, if the source codec does its job properly, the data stream coming out of it is both considerably smaller and also looks, statistically, quite random. Some algorithms for doing this are very generic in nature and do not take into account the nature of the information, only its current representation; file compression programs commonly fall into this category. Other algorithms rely

heavily on knowing the nature of the information in order to achieve much higher degrees of compression on data of that type (and typically much poorer compression on other types). Examples of these are compressed file formats such as MP3 for audio streams, JPEG for images, and MPEG for video data. The need for high degrees of compression is particularly important in applications such as the cellular telephone network. Thus, cell phones use highly refined and optimized audio codecs to convert the raw audio data streams from the microphone into tightly compressed data packets that convey just enough information for the receiving phone to convert them back into an audio signal with enough fidelity, hopefully, to be recognizable and intelligible. In fact, applications such as this are what is normally meant when someone refers to a "codec".

Moving on to the "security codec" – which is more commonly called an encryption module or even just an encrypter on the transmit side and a decrypter on the other – its obvious purpose is to transform the input data stream into something that looks like highly random nonsense and that requires special information, known as a "*key*" to convert back into something meaningful. The security codec's output will probably have the same number of data bits as its input, and thus would be expected to have little impact on the number of bits that have to be transmitted.

In turning to the "*channel codec*", we also make a transition in the immediate goals of the codecs. Up to this point, the goal has been to remove redundancy and variation in order to minimize the number of bits that must be transmitted and make the data more secure. Now, we will begin intentionally introducing redundancy and variation, in very carefully constructed ways, in order to improve the reliability of that transmission in the presence of noise, either natural or malicious. In what initially might seem to be a paradox, the purpose for doing this is to further reduce the total number of bits that have to be transmitted. How this is possible is that, in a noisy channel, some of the data bits will invariably become corrupted or lost

entirely. If the data has been optimally represented (and especially if it is encrypted) then the loss or corruption of a single bit of data results in the total corruption of the entire message. The channel coding is designed to make it possible to detect and correct this corruption, up to a point, and reduce the amount of data that has to be retransmitted.

To illustrate a simple example of channel coding, consider the case where there is no source or security coding at all and we simply transmit the data as straight ASCII characters. Because the most significant bit in each ASCII byte is zero, the receiver will have little problem locking onto the byte boundaries, even if this bit occasionally gets flipped. So although that bit conveys no information and would be completely extraneous in a noise-free system, the mere fact that it is entirely redundant allows the receiver to synchronize to the data stream in the presence of noise. Now assume that each byte received were displayed on a screen or written to a text file and examined. Most people would have little difficulty extracting the information content even if a fairly significant fraction, say 10%, were corrupted. The reason that this would be possible is because the data inherently contained a lot of redundant information that a human reader could draw upon in order to reconstruct most, if not all, of the corrupted or missing information. Furthermore, even if reconstruction weren't possible, it would be all-but-certain that that the human reader would detect that the received message was corrupted beyond recovery and request the sender to retransmit. On the other hand, it would be very difficult, if not impossible, to develop a computer program that could perform this same recovery task at anywhere near the level of noise that a human could perform. So let's consider the two types of recovery described here. The first, relying on the leading bit of each character being zero and using that to permit a simple logic circuit to synchronize to a noisy data stream, represents redundant information that is not there because of any fundamental feature of the the English language, but rather because of a coincidence in how characters are represented in

a computer. Being able to reconstruct corrupted characters, on the other hand, is an example of humans relying on redundancy that exists in the language itself. It is perhaps not coincidental that human languages, both written and spoken, have such redundancies and that humans are able to so naturally exploit them, such as when trying to make out a conversation in a noisy room. It would not be surprising if both evolved hand-in-hand. Similarly, it is not surprising that if we want to permit a computer to perform these same tasks, namely detecting that corruption has occurred or, even better, correcting a significant amount of it, that the redundant information needed would best come from algorithms specifically designed for that purpose since, at least presently, letting them evolve naturally is not a viable alternative.

Coming back to the channel codec, its purpose is to encode the data and add redundant information so that the receiving codec can recover the correct data even in the presence of some maximum tolerable amount of noise. The term *channel* coding is used because the best algorithm is one that leverages knowledge about the types of noise and its likely effects in the transmission channel used to convey data from sender to receiver. For instance, the manner in which noise from other cell phones impacts a signal between a moving vehicle and a cell tower is very different from the manner in which noise from the sun impacts a signal returning to Earth from a deep space probe such as Voyager. Similarly, the ability, or lack thereof, to request a transmission impacts the decision on how much redundant information to add; a cell phone user can ask the other person to repeat what they just said or even reestablish the connection if necessary while data transmitted from Voyager represent a one-time opportunity since the data buffers are overwritten long before any retransmit request could possibly be received.

The coding algorithms used by channel codecs are generally described as "*error-detecting*" or "*error-correcting*" codes. Examples range from simple parity bits to a broad range of linear and non-linear codes including families such as BCH and Reed-

Solomon [9, 10]. Furthermore, codes can be streaming, such as convolution codes, or packetized, such as block codes. Regardless of the details of a specific code, each one introduces additional bits into the data and is generally described by a "*code rate*", R_C, which is defined as the ratio of the number of input bits to the number of output bits.

Thus, a code rate of "4/7" means that for each four bits presented to the codec, seven bits are output. Since we will be focussing on a packetized coding system, we will only consider those. At this stage, it is sufficient to note that a block code generally accepts an L-bit message and produces an C-bit codeword, with $C \geq L$. Therefore the code rate for a block code is

$$R_C = \frac{L}{C} \tag{4.1}$$

As would be expected, a general goal is to use the highest rate code that will accomplish the error detection/correction objectives.

Looking again at our example, let's assume that the channel codec is using a 1/2 rate code and consumes data from its input eight bits at a time and produces 16-bit codewords at its output. The average entropy would therefore be cut in half to 1/4 bit of information per codeword bit and the variability would likely be increased considerably. For this example, let's assume that the standard deviation is increased to 1/100.

Finally, we come to the "*spectrum codec*", more commonly known as the spread spectrum modulator. There are several general reasons why a system might choose to use spread spectrum. One would be to make the system tolerant of very poor signal-to-noise ratios (SNR), such as is the case in the Global Positioning System (GPS). Another is to permit multiple systems to share the same spectrum allocation, as is the case with many modern systems ranging from some cellular technologies

to many wireless home automation devices. A third, which is the one we are most interested in, is to make the signal resistant to intentionally crafted noise broadcast by an adversary whose purpose is to disrupt communications. In each case, the idea is to take a signal that could normally be transmitted in a particular bandwidth, W_{NB}, as a narrowband signal and instead use a considerably larger bandwidth, W_{SS}, in order to give it a level of immunity to the noise it is expected to encounter. In practice, the ratio of these two bandwidths could range anywhere from about ten to tens of thousands. For instance, some radio modems in RF Innovation's Piccolo line [11] have as little as few as 5Hz per bps (bits per second) (bandwidth/data rate is another common metric) while civilian GPS [12] uses a 1.023MHz bandwidth to sustain a 50bps data rate, or more than 20kHz/bps.

Like the channel codec, the spectrum codec that works best is one that is chosen with the specific type of expected noise in mind. The performance of a spread spectrum system is properly defined as the ratio of the increase in performance of the system with spectrum spreading to the performance without it. This ratio is known as the "*processing gain*", G_P. Unfortunately, this definition requires the additional definition of what is meant by "performance", which might be quite different for different objectives even with the same system. A somewhat looser definition that is usually still a reasonable approximation is the ratio of the spread transmission bandwidth to the underlying data rate. When focussing just on the spectrum codec, the data rate would refer to the bit rate being presented to its input, since it seldom has any awareness of either the existence or the impact of prior coding blocks. On the other hand, when comparing overall systems, such as candidate systems intended to transmit video data between two stations, a more holistic definition might be used that would compare, perhaps, the ratio of the frame rates that could be supported across the link taking into account everything from the video compression to the need to occasionally retransmit data that couldn't be corrected on prior attempts.

For completeness, we will talk briefly about the "*modem*", or the "modulator/demodulator". Many spread spectrum systems use separate data modulators and spreading modulators. Furthermore, some systems modulate the data prior to performing the spreading and others do it afterward. For many purposes, these are theoretically equivalent ways to accomplish the same ends and the specific choices are often driven by practical design considerations. The specific choice reflected in Figure 4.1 represents the fact that this dissertation focusses on a system that intrinsically combines the channel and spectrum codecs into a single block.

4.2 Channel Coding Meets Spectrum Coding

Summarizing to this point, most modern communication systems employ several different types of coding. First, the information source may have significant redundancy in the data it generates and a source codec removes as much of this as possible to minimize the number of bits that must be transmitted. There might also be a security codec, typically a cryptographic module of some sort, intended to scramble the data so that its content is extremely difficult, if not impossible, for a third party to divine should they intercept it. The data might then be passed through a channel codec that inserts redundancy back into the data, except that this redundancy is specifically intended to permit the receiving codec to correct at least some of the errors that the signal experiences during transmission. Finally, a spread spectrum system applies a spreading code in the spread spectrum modulator (the spectrum codec) to smear the information signal over a much larger portion of the spectrum making it harder to detect and/or jam. In comparing systems to each other, or to some benchmark, the processing gain is the ratio of the relative performance with and without a particular enhancement.

For our purposes, a reasonable metric is the ratio of the transmitted bandwidth to the data rate presented to the channel codec, R. Thus, our processing gain is defined as

$$G_P = \frac{W_{SS}}{R} \tag{4.2}$$

This definition of processing gain is useful for us because we wish to focus on a new type of codec, called a "*concurrent codec*", that is envisioned as performing the functions of both the channel codec and the spread spectrum codec in one processing operation. While some aspects of this new coding algorithm are clearly more akin to channel coding and aspects to spectrum coding, the two operations are combined in such a way that it is not possible to separate them into distinct, independent blocks. Thus, for comparison purposes, we want to compare the performance of a concurrent codec to the combined performance of a traditional channel codec and spread spectrum codec while keeping all else equal.

4.3 Analogy to Public-Key Cryptography

This situation is not without precedent and an understanding of a similar historical problem is useful for framing this discussion. Until the advent of asymmetric cryptography in the early 1970s[1] [13, 14, 15, 16], it was nearly universally accepted that secure encrypted communications required the prior exchange of keys via a secure unencrypted channel. For entities such as governments and large corporations, for whom the overhead and expense of distributing keys was an acceptable cost for securing their internal communications, such a limitation was merely a significant hindrance - these organizations had the resources to deal with the distribution problem, though in large part this was because the number of parties to whom keys had to be

[1] Asymmetric cryptography had actually been developed a decade or so earlier by the National Security Agency and other intelligence agencies, but this was not made known until long after it was independently developed and put into the public literature

distributed was relatively small and manageable. In contrast, however, the cost and difficulty of securing large scale personal and consumer transactions via the secure distribution of keys quickly approaches the unimaginable.

Attempts to extend cryptographic protection to larger networks has included a variety of strategies, but all have required prior secure communications - if not with the other party than with a trusted third party. Perhaps the most evolved such system is Kerberos [17, 18], developed originally at MIT during the early 1980's. Kerberos permits secure communications between two parties who have no prior knowledge of each other over an insecure network by acting as a central clearing house for session encryption keys. However, the system requires that each party have a prior relationship (and shared secret) with the Kerberos server and that the Kerberos server be available at the beginning of the communication session. If either of those requirements is not met, secure communications cannot commence.

The symmetric key distribution problem therefore posed an enormous hurdle for electronic commerce. Asymmetric cryptography fundamentally changed this situation. Instead of both parties sharing the same key, asymmetric algorithms use two different, but related keys. These cryptographic algorithms use "asymmetric keys" in which each party in the exchange uses different, but related, keys [7]. In essence, one party generates both keys and whatever is encrypted with one key can only be decrypted with the other. That party then keeps one key private and makes the other key public. Using the public key, anyone can encrypt a message that only the holder of the private key can decrypt (provided there is no feasible means of deriving the private key from the public key). Notice that there is still a secret involved and there is still sharing involved; however there is no shared secret. One key, the "private key," must still be closely held and safeguarded, but never needs to be distributed; the other key, the "public key," may need to be widely distributed but does not need to be safeguarded. By splitting the key management problem this way, highly scal-

able solutions can be developed; in fact, without asymmetric keys, e-commerce could not function at anything approaching the scale that it presently does.

In particular, asymmetric makes possible the still-expanding Public Key Infrastructure (PKI) for the distribution of public keys in such a way that they can be used to authenticate message traffic as well. Thus it is possible to ensure the confidentiality, integrity, and authenticity of data transferred between parties having no prior relationship [19, 20, 21, 22] without requiring a solution to the scalability limits of symmetric key distribution. Even without PKI, it is possible to securely exchange information over an insecure network using, for instance, the Diffie-Hellman Key Exchange algorithm - what is lost is the ability to authenticate the other party.

It would be incorrect, however, to conclude that asymmetric algorithms supplanted symmetric ones. Present asymmetric algorithms are very computationally intensive and it is highly likely that symmetric algorithms will always be the more efficient of the two. This is simply due to the significantly greater constraints placed on the behavior of an asymmetric algorithm compared to its symmetric brethren. As a result, the data throughput for an asymmetrically encrypted channel is generally much lower - typically one or two orders of magnitude - than for a symmetrically encrypted one. To overcome this penalty while maintaining the independence from shared secrets, a hybrid approach is typically employed wherein an asymmetric channel is used to establish communication, but only for the purpose of exchanging short-lived session keys that are used to set up an appropriate symmetric channel. In modern communications, the overwhelming use of asymmetric algorithms, such as Diffie-Hellman key exchange and RSA, are for the secure exchange of symmetric session keys between parties while symmetric ciphers such as 3DES[2] and AES[3] continue to protect the bulk of the traffic. In practical terms, symmetric keys are still used and

[2]Generally referred to as Triple-DES, 3DES is a three-round variant of the Data Encryption Standard effectively providing a 112-bit key strength.

[3]Advanced Encryption Standard - The nominal successor to DES which provides several key-length options with the most common being 256 bits

the associated key management issues still exist, but they have been partitioned to a hyperfine degree. Instead of a central authority generating keys that many parties will use for an extended period of time and devising way to securely transfer them, they are generated on the fly, distributed to only two parties, only exist for the duration of a single session, and are securely transmitted over an insecure channel as part of the asymmetric handshaking protocol.

Unfortunately, while asymmetric public/private key approaches permit information to be exchanged securely over an insecure channel, they do require that such a channel, even if insecure, exist and be functional (i.e., unjammed). A means of using asymmetric keys to achieve jam resistance in the physical layer has yet to be devised. In other words, asymmetric keys could be used as TEKs, but not TSKs. Despite this, the world of omnidirectional wireless communications is quite analogous to that previously faced by the information assurance world. Modulation schemes such as spread spectrum are almost direct counterparts to symmetric ciphers in several central respects: (1) efficient, high-bandwidth, jam-resistant communications are possible, provided a shared secret can be successfully exchanged; (2) shared secret distribution is a limiting factor in the large scale deployment of such jam-resistant channels; (3) what is needed is an analogous means of exchanging keys via an insecure channel; and (4) significant penalties in data throughput and processing cost can be tolerated while performing the key exchange.

4.4 Requirements for Unkeyed Jam Resistance

Put simply, our problem is to devise a means of transferring a message from a sender to a receiver via omnidirectional broadcast in the face of significant hostile jamming even when the jammer possesses any and all knowledge that is common to both legitimate parties. Given those constraints, we need to determine what actions are

available to the jammer and what characteristics our system must possess to resist those actions.

Restricting the discussion to omnidirectional wireless communications, the only means currently available of providing significant resistance to hostile jamming is the use of spread spectrum techniques, such as frequency hopping and direct sequence. However, as previously mentioned, all traditional forms of spread spectrum achieve resistance through the use of a shared-secret spreading code, such as the hop sequence used by FHSS or the chip sequence used by DSSS. More explicitly, in light of the prior discussion, the spreading code forms a "symmetric key" since the same code must be known to both sender and receiver. Under the stipulations of our problem statement, we must therefore assume that the jammer has somehow obtained the spreading code. With this information, the jammer can generate a valid signal that will interfere with the legitimate signal so as to increase the jamming vulnerability to a level comparable to a narrowband transmission.

Before considering what is required to make an unkeyed system jam resistant, let's first acknowledge than any unkeyed system has to give up low probability of detection (LPD) since anyone can now detect the signal as easily as the legitimate receiver. For systems in which LPD (separate from jam resistance) is critical, an unkeyed jam-resistant system would not be an option, except perhaps in a hybrid system in which a very short unkeyed session was used to exchange session TSKs for the remainder of the conversation, using either a Public Key Infrastructure (PKI) or something similar to a Diffie-Helman key exchange [8, 23].

One way in which to view this situation is that the spreading code effectively defines a dynamically shifting narrowband channel through which a message signal is tunneled. Without knowing the code, the best the jammer can generally do is craft a signal that will, statistically, inject enough noise into the desired tunnel to produce the desired adverse affect. But with the spreading code, they can make sure

their signal is injected entirely into the correct tunnel. As is generally the assumption in the communications community regarding "smart jammers", we will grant the adversary some rather unrealistic capabilities in order to determine the worst case bounds. In particular, the usual assumption is that the adversary knows everything about your system (except the spreading code, which we are also granting them knowledge of) and therefore can generate a perfectly valid signal that is optimally aligned at the receiver to interfere with the legitimate signal. Further, we assume that they know the legitimate signal-to-noise ratio (SNR) at the receiver front end. Given these capabilities, the most effective jamming signal is going to be one (or many) valid signals overlaid on the legitimate signal such that they arrive at the receiver at the same time. Given that almost all receivers are designed to lock onto a single valid waveform and reject everything else as channel noise, this situation will almost certainly jam the channel completely because, if nothing else, the receiver will be faced with a decision of which of the many potential signals to lock onto and will almost certainly choose wrong. But even this is too optimistic – with the spreading code at hand, the jammer can simply take whatever narrowband waveform would have been most effective against a non-spread version of the system and then modulate it using the same spreading code as the legitimate signal. Thus the jammer will most likely be able to drive the bit error rate (BER) toward unrecoverable levels using power levels comparable to that of the legitimate signal.

With all of this in mind, let's consider the requirements for achieving jam resistance (in an omnidirectional signal) without any secret keys at all. To be more specific, what challenges would a spread spectrum system have to overcome if the spreading code were publicly disclosed? As is always the case, it is assumed that a hostile adversary will fashion the attack signal, within their own capabilities and limitations, that minimizes the ability of the legitimate receiver to recover information from the received signal. We've already established that arguably the most effective

jamming signal would simply be other valid signals present concurrently with the legitimate signal. Thus, we will refer to channels in which multiple valid signals are potentially present as a "concurrent channel" and accept that any unkeyed spread spectrum system must contend with such a channel.

However, while extracting information modulated onto a signal that has been badly corrupted by an attacker's overlying valid signal is a daunting task, merely detecting that such a signal is present is a much easier undertaking. With this in mind, a system that encodes information only by the presence of a specific set of signals (which will be called "marks") has the potential to allow information to be conveyed in a concurrent channel. This is because it is no longer sufficient for the attacker to simply distort the legitimate signal until no information can be demodulated from it; instead, they must successfully prevent the presence of the signal itself from even being detected. In theory, it is possible to construct an attack signal that could cancel out some types of legitimate signals at the receiver. In practice, however, just canceling a single-tone signal this way is extremely difficult since even small errors in frequency, phase, or amplitude are likely to leave a detectable residue.

While a seemingly subtle distinction, it is actually critical to note that the information must be encoded solely in the presence of a particular set of marks (however distorted those marks might become). The information encoding cannot rely on the absence of specific marks (or, in other words, the presence of a particular set of "spaces"). How this is achieved, and the limits of doing so, will become apparent when the BBC algorithm is discussed later. Because the legitimate marks need to be detected but we can tolerate many spaces being lost, the ideal channel for a concurrent code spread spectrum system is an OR-channel. In this type of channel, a mark is detected by the receiver if any signal source, friendly or hostile, delivers a mark, while a space is declared only when all signal sources, friendly and hostile, contribute spaces. In practice, like any other ideal channel model, an OR-channel cannot be

perfectly realized. However, as will be shown, even very simple transmission schemes can provide quite reasonable approximations in practice.

We can now make a key observation about the nature of the realizable communication channel that we want for an unkeyed form of spread spectrum such as this to work. We will limit the discussion to a binary channel in which the two symbols are a "mark" and a "space." The receiver's primary objective is to correctly detect all marks that are transmitted. However, it does not have to correctly detect spaces; for the basic BBC encoding analyzed in this paper, only about half must be detected as spaces (overwhelmingly, any half will do) [24, 25, 26]. Consequently, we want to use a channel in which the probability of detecting a legitimate mark is very high; fortunately, we can tolerate a significant probability of falsely declaring a mark when a space should have been detected. A channel such as this is highly asymmetric and very atypical in modern communications, so our analysis will begin from the basic properties of such channels.

Chapter 5

Literature Search

The literature search for this project focussed on two areas: (1) What work has been done to enable jam-resistant communications in omnidirectional systems in the absence of a symmetric key, and (2) what is the existing body of knowledge relating to the behavior of binary sequences combined via a bitwise-OR.

5.1 Relevant Work on Jam Resistance

Since one of the chief claims of this research is that it is the first (successful) effort to achieve significant jam resistance in an omnidirectional system in the absence of an uncompromised symmetric key, this part of the search equated to attempting to prove a negative. Numerous papers were found that dealt with the jam resistance of communication systems. For instance, in the case of spread spectrum systems, much attention has been given to performance against jamming and various mitigation strategies both generally [27, 28, 29] and in specific subfamilies including direct sequence [30, 31, 32, 33, 34, 35], frequency hopping [30, 36], and pulse-position systems [37, 38, 39]. However, all employ either directionality or symmetric keys. Similarly, examples of fielded systems that claim significant jam resistance were also not hard to find, but again all of them rely on directionality and/or symmetric keys.

Given the growing importance of jam resistance in general, it is unlikely that all substantive work in this area would have evaded the major research publications and databases such as CiteSeer and IEEExplore.

5.2 Relevant Work on Superimposed Codes

In contrast, exploration of the behavior of binary sequences that have been combined using a bitwise-OR enjoys a rich history over the past six decades. The topic has been studied by several highly disparate communities, ranging from pure math theorists to practical information retrieval system designers. These groups appear to have had little interaction with each other in this topic area and, as a result, a plethora of terminology exists and many different words and phrases are used for the same, or at least similar, concepts. The most widespread term for the bitwise combination of messages is *superimposed codes* [40, 41, 42, 43] and many other terms [44, 45, 46, 47, 48, 49, 50, 51] have followings in various communities. Many of these codes have been constrained in various ways to improve specific performance parameters [52, 53, 54, 55, 56, 57, 58, 59, 60], but none have focussed on providing an efficient means of decoding large collections of codewords that have been bitwise-OR'ed together. Typically, these codes are for a fairly small codebook, such as one codeword per user in a system or cell-culture in a biology study, so it's more likely to be on the order of thousands (or perhaps millions) of codewords rather than exponentially large codebooks such as 2^{1000} [61, 62, 63, 64, 65].

One indication of the overall significance of superimposed codes is that the U.S. National Institute of Standards and Technology (NIST) maintains a definition for a superimposed code: "A set of bit vectors such that no vector is a subset of a bitwise-OR of a small number of the others" [66].

In keeping with tradition, yet another name is introduced here, namely *concurrent codes*. However, this is justified because concurrent codes are not just superimposed codes by another name; concurrent codes are a subset of superimposed codes possessing a property that has previously eluded other researchers. Specifically, concurrent codes are the subset of superimposed codes that can be efficiently decoded (preferably in linear time).

Actually, concurrent codes are a subset of a slight generalization of superimposed codes. Superimposed codes guarantee that the combination of any number of codewords, up to some maximum, will contain no other codewords at all. Concurrent codes, on the other hand, only provides a probabilistic expectation value of the number of additional codewords contained that can be made arbitrarily small. This evolution in the definition is similar to what has happened historically in error correcting codes. The early codes had guarantees that it could recover any codeword with up to a certain number of errors. But modern codes guarantee only a high probability of recovering the codeword. So "concurrent codes" can be thought of as "superimposed codes" that are constrained to be efficient, but allowed to be probabilistic.

For simplicity, it is useful here to define a single vocabulary that will be used throughout this document and then use that vocabulary, wherever possible, even when discussing the work of a particular field that prefers a different nomenclature. Translating the NIST definition into this terminology, the "bit vectors" are "*codewords*" and the "bitwise-OR of a small number of (them)" is a "*packet*". If a codeword is a subset of a packet, it is said to be "*covered*" by or "*contained*" in the packet. Hence, the NIST definition of a superimposed code in our terminology would be, "A set of codewords such that no codeword is contained in a packet composed of a small set of other codewords."

What has become fairly apparent after reviewing the literature is that nearly all of the existing work has proceeded from a central premise - that the set of codewords

is relatively small and, where necessary, fully enumerated. It is a simple matter to determine if a specific codeword is contained in a packet, whereas identifying all of the codewords that are contained in a packet is a much more challenging task. However, if the set of codewords is sufficiently small, it can be accomplished in an acceptable amount of time via exhaustive search. In fact, the inability to extract codewords from a packet except via exhaustive search has been touted as a benefit for some applications.

That so much attention has been devoted to superimposed codes without a corresponding amount of effort spent attempting to devise a means to efficiently decode packets containing them may seem odd at first. However, a brief overview of the types of applications they arose from sheds quite a bit of light on this apparent oddity. The earliest use of what would become superimposed codes appears to have originated by Robert Dorfman and David Rosenblatt in the early days of the United States involvement in World War II [67]. Faced with a need to screen millions of service inductees in order to identify a few thousands of cases of syphilis, Dorfman and Rosenblatt, economists by training, devised a means of significantly reducing the total number of tests that must be performed by pooling blood samples and testing only the pools. If, they asserted, the initial screening was performed by grouping the blood of five inductees into a single test sample, then if the test returned negative there was no need to perform further tests on any of those five people. If, however, the test was positive then the individual(s) infected could be identified by retesting just those five people (using blood held back for that purpose). Since most of the pools would return negative results, the total number of tests performed would only be a little over one-fifth of the number needed for individual testing. As it turned out, this method of screening for syphilis was never put into practice for a variety of reasons; however Dorfman published his proposal [68] and hence was started the field of "group testing".

One of the most frequently cited works in the field of superimposed codes is the 1964 paper by Kautz and Singleton [40], Many papers confer upon them the honor of inventing superimposed codes. However, Kautz and Singleton themselves, in that initial work, refer to the 1958 work of Schultz [69]. Schultz's work involved the effort to categorize and cross reference the unprecedented amount of technical literature developed during World War II. A variety of schemes had been proposed, some practical and some not, of performing data searches on large volumes of information. One such method involved taking a large index card with the reference information for publication and punching a line of small holes along one edge. Then the publication would be classified according to a list of categories and for each category in which it belonged a subset of the holes assigned to that category would be turned into slots extending completely to the card edge. Each card would therefore end up with a set of holes and slots which served to identify all of the categories to which the publication belonged. However, since there were many more categories than there were holes, it was necessary for the same slot to be used for multiple categories. When a search needed to be performed, a stack of cards would be assembled and small rods would be passed through the holes associated with category of interest. Then the rods would be held up and any cards that had slots at the locations of all the rods would drop out of the stack. Any card which did not drop would be known to not belong to the category being tested, although it was possible that some "false-drops" would occur meaning that some cards might drop that did not belong to the category. As computers, then very much in their early days, were developed, this manual index-card based system lent itself quite naturally to the migration to computer-read punched cards. Much of the subsequent development of superimposed codes arose from devising various ways to minimize the probability of false drops while maximizing the number of categories that could be encoded using a limited number of holes.

As (relatively) modern computers continued to find new applications, superimposed codes gained interest in a variety of venues, but nearly all had at their core the goal of membership testing within a group. Perhaps the archetypical example, generally associated with Bloom filters [44], is the notion of a dictionary. In this example each word in the dictionary is encoded into a codeword and then all of the codewords are combined into a packet. At this point, a spell check can be performed by taking each word in the document, encoding it, and testing if the codeword is covered by the packet. If it isn't, then it is guaranteed that the word is not contained in the dictionary. If it is, then the word probably is in the dictionary, but there is a slight chance that it is not. Bloom and others spent considerable effort devising codes that minimize the probability of false hits, but the premise has always remained that the areas of use would continue to involve only membership testing. A consequence of this is that, while it is easy to test if a particular word is in the dictionary, it is impossible to take the dictionary packet and generate a list of all the words in the dictionary. This one-way behavior has been offered as being a feature in the sense of the following example: All of the social security numbers of all of the people in a particular group, say people that the government owes money to, are encoded and combined into a packet. The packet is now made public and any person can easily test whether the government owes them money. However, it is impossible to generate the list of social security numbers contained in the packet except by exhaustive search (which, today of course, would not be much of a hurdle, but the idea is there).

The inability to efficiently decode an arbitrary packet in the face of an exponentially large set of codewords has not been of major consequence for most of the work described in the literature. Most of it has been concerned with determining the likelihood that a given codeword will be incorrectly determined to be contained within a packet even when it was not one of the codewords used to form the packet. This is not to say that the desirability of efficiently decoding packets drawn from very

large codebooks has never been recognized. However, as recently as 2003 Cormode and Muthukrishnan concluded that no method short of exhaustive search existed for decoding [70]. It's not known if these authors were aware of work published in 1988 that did present a relatively efficient (i.e., polynomial time) algorithm for decoding packets containing fairly small numbers of codewords [71, 72, 73, 74, 75]. Even if such algorithms were known to them, it's quite conceivable that they would have been deemed inadequate since the algorithms were of complexity $O(n^3)$ or $O(n^4)$, hence packets containing 1000 codewords could have required as many as a trillion iterations of the algorithm. In contrast to this, the BBC algorithm developed here decodes packets in linear time, $O(n)$.

Chapter 6

Concurrent Coding Theory

This section focuses exclusively on the encoding and decoding algorithms and does not concern itself with transmission issues such as noise or timing considerations; those will be covered in a later section. The discussion will be focussed on a concurrent codec, whose requirements, capabilities, and characteristics will be developed incrementally. By the end of this section, the complete Standard BBC codec algorithms will be have been covered, including their quantitative performance expectations. Included in the discussions are several enhancements to the Standard BBC codec.

6.1 Concurrent Codecs 101

A codec (coder/decoder) has two data paths, one for data to be transmitted and one for data being received. The transmission side of a concurrent codec is largely indistinguishable from a conventional channel codec in that it accepts a data stream and encodes it into one or more codewords which are then presented to the physical layer block of the transmitter for modulation and eventual broadcast.

Existing concurrent codecs are intrinsically packet based, thus an entire L-bit message (which is nothing more than a fixed-width collection of information bits and overhead bits) is presented to the codec which then transforms it into a single C-bit

codeword. The ratio

$$F = \frac{C}{L} \tag{6.1}$$

is the expansion factor and is the single most important parameter in establishing the jam resistance of the resulting system. As has already been mentioned, the concurrency efficiency, η, of a concurrent codec serves as a rough analogy to the code rate of a traditional channel codec. The expansion factor similarly is a rough analogy to the spreading ratio discussed in the context of traditional spread spectrum modulators since it is directly proportional to the increase in bandwidth associated with using CCSS (or, equivalently, the decrease in data rate suffered while keeping the transmission bandwidth constant).

Likewise, the receiving side of a concurrent codec has most of the same goals as conventional channel codecs, namely to accept a bitstream from the receiver's physical layer block, identify codewords that are contained in it, and decode them to reassemble the original data stream and pass it to the next higher block in the architecture. However, there are two fundamental differences in how this is done compared to most conventional channel codecs. First, conventional codecs are not concerned with any aspect of spectrum spreading, whereas a concurrent codec performs this task as well as channel coding. Second, a traditional channel codec assumes the presence of a single, possibly corrupted, codeword (at a time) and tries to recover the original data from it, or at least detect that corruption has occurred. While codecs based on error-correcting codes can be made quite robust against non-intelligent noise, they are easily defeated with intelligently crafted noise. In essence, error-correcting codes are too clever for their own good. What is needed is a codec that makes no attempt to figure out what the true original data was but, instead, simply generates a list of all possible data segments that are consistent with the received packet regardless of the amount of corruption. Such a codec is defined as a *concurrent codec*.

The model that will be used assumes that an arbitrary chunk of data is to be transmitted from Node A to Node B. The data will be partitioned into a series of messages that will contain routing and reconstruction information along with some portion of the data. The result will be a sequence of messages that will be presented to the transmitting codec. The codec will then encode each message into a codeword and assemble them into a sequence of one or more packets that will be broadcast. The receiver will detect the transmission and produce a continuous stream of bits within which are contained (hopefully) the transmitted packets. A packet is nothing more than a segment of the received bitstream large enough to encompass one codeword. Each packet may contain multiple codewords and successive packets may overlap. The receiving codec will extract any and all codewords that are consistent with each packet it examines, decode the codewords into the corresponding messages, and pass along the resulting message list. The next layer will authenticate each message and pass along only those that remain valid to a final block that will extract the data from them and reassemble it. Here, of course, we are only concerned with that portion of the processing that is performed by the codec itself.

In the case of a transmission channel with symmetric error probabilities, every position in the codeword has the same probability of containing an error and thus every received packet is consistent with every possible codeword (because of the 'regardless of the amount of corruption' clause) and the resulting message list would always contain every conceivable message. However, with a sufficiently asymmetric channel, only those messages whose codewords are *covered* by the received packet are included in the list. To be covered, a codeword must have all of its marks in the packet (this absolute constraint will be relaxed later when codec enhancements are considered).

For now, we will assume that the channel is sufficiently asymmetric such that there is effectively zero probability of a mark (a '1' bit) being turned into a space (a '0' bit)

but that there is a fair likelihood of spaces being corrupted by noise and turned into marks. Under this assumption corruption can only turn spaces into marks - marks that are actually transmitted will be received (this assumption will be somewhat relaxed later).

If the attacker can force us to receive a packet of all marks then the channel is completely jammed. In practice, the channel becomes effectively jammed if the attacker forces us to receive a packet that has too high a percentage of marks. Hence our anti-jamming strategy can be concisely summarized as needing to be able to operate in the presence of as many marks as possible while simultaneously making it too expensive for the attacker to force us to operate any closer.

For illustration purposes, let's consider a toy example having four-bit messages with an *expansion* of eight (i.e., the codewords are eight times as long as the message, thirty-two bits in this example) and a *mark factor* of one (i.e., there is one mark in the codeword per message bit, four marks per codeword in this example). Each of the sixteen possible messages maps to a unique codeword. The mapping is performed by the encoder portion of a codec using some algorithm. For now, it is sufficient to simply list the messages and the corresponding codewords, the algorithm used is irrelevant. The resulting codebook is shown in Table 6.1.

Because all codewords are unique, any time a single message is encoded and transmitted only that message will be decoded at the other end unless some form of corruption occurs. As marks are added to the packet, either by noise or an attacker, eventually additional codewords will be covered by the packet and the codec will add the corresponding messages to the list. Since it is assumed that the attacker has access to the same information as the legitimate sender, they will always be able to force a concurrent codec to add additional messages to the list while expending no

Table 6.1: Toy codebook.

#	Message	Codeword
0	0000	00010000 00100000 00000010 01000000
1	0001	00000000 00010000 00001010 00010000
2	0010	01000000 01000000 01001000 00000000
3	0011	00001000 00010000 00000000 10000001
4	0100	00000000 10000010 00010000 00010000
5	0101	00010000 00000001 10000000 00000100
6	0110	00100000 00010000 00010010 01000000
7	0111	00011000 00001000 00000000 00100000
8	1000	00000000 00100100 00000000 00100100
9	1001	00010010 00010000 00000100 00000000
10	1010	00000000 00000100 00000000 00101010
11	1011	00100100 01000000 00010000 00000000
12	1100	00010000 00001000 00000001 00001000
13	1101	10000001 00000100 00010000 00000000
14	1110	00010100 00100000 00100000 00000000
15	1111	00001100 00000000 00001000 00000010

more energy per message than was required by the legitimate sender, simply by using the same process.

Consider a packet formed by the legitimate sender transmitting message 3 and the attacker sending messages 6, 9, and 12. Assuming the codewords are perfectly aligned results in the packet shown in Table 6.2. When these messages are decoded (by brute force examination of each codeword in the codebook in this case) it is found that the packet covers messages 3, 6, 9, and 12 and, hence, these are the messages in the codec's output list.

In contrast, consider a packet formed using the legitimate message 2 and attack messages 6 and 14 (see Table 6.3). Because of the interaction between marks contributed by the individual codewords, the resulting packet covers additional messages, namely 0 and 11, that were not part of the generating list.

Table 6.2: Toy example without hallucinations.

#	Message	Codeword
Generating Message List		
3	0011	00001000 00010000 00000000 10000001
6	0110	00100000 00010000 00010010 01000000
9	1001	00010010 00010000 00000100 00000000
12	1100	00010000 00001000 00000001 00001000
Packet		00111010 00011000 00010111 11001001
Decoded Message List		
3	0011	00001000 00010000 00000000 10000001
6	0110	00100000 00010000 00010010 01000000
9	1001	00010010 00010000 00000100 00000000
12	1100	00010000 00001000 00000001 00001000

As the previous examples illustrate, a concurrent codec can produce output messages in addition to those sent by the genuine sender. These additional messages can come about in one of two ways - either because the attacker explicitly inserts them or because of interactions between messages (and other noise) in the packet. Intentionally inserted codewords are called *rogues* to distinguish them from those transmitted by the legitimate sender. Spontaneously appearing codewords are called *hallucinations* because they don't really exist but, like any good hallucination, are hard to distinguish from the real thing.

From the receiver's standpoint the distinction between rogues and hallucinations is largely, though not totally, superfluous. The concurrent codec must extract hallucinations as well as rogues but, since hallucinations are essentially random messages, the application may be able to identify many of them as invalid without expending a significant amount of effort. Whether this distinction matters in the end depends on where the processing bottleneck occurs - the concurrent codec or the application. However, from the attacker's perspective the distinction is critical - they have to pay for rogues but hallucinations are free. If they can identify a set of rogue messages that produce a significant number of hallucinations, then they can greatly reduce their av-

Table 6.3: Toy example with hallucinations

#	Message	Codeword
Generating Message List		
2	0010	01000000 01000000 01001000 00000000
6	0110	00100000 00010000 00010010 01000000
14	1110	00010100 00100000 00100000 00000000
Packet		01110100 01110000 01111010 01000000
Decoded Message List		
0	0000	00010000 00100000 00000010 01000000
2	0010	01000000 01000000 01001000 00000000
6	0110	00100000 00010000 00010010 01000000
11	1011	00100100 01000000 00010000 00000000
14	1110	00010100 00100000 00100000 00000000

erage energy per message and hence have a much higher probability of jamming the channel. Their ultimate goal is to identify small sets of rogue marks (as opposed to rogue messages) that are highly hallucinogenic.

6.2 Basic Requirements of a Concurrent Codec

To be acceptable for use in a practical implementation, a concurrent codec must be jam resistant, efficient, and scalable. Clearly jam resistance is the most important consideration, for without that the whole exercise is moot. But even an ideal system is worthless if it cannot be fielded in practice. Fortunately, the jam resistance does not have to be perfect, it merely has to be good enough. This permits the designer to trade off jam resistance in favor of attaining acceptable efficiency and scalability. Efficiency will almost certainly be important to most applications. While each particular application will be different, in general the implementation will have to be designed with processing, energy, space, and cost efficiencies in mind. Most applications are likely to be real-time in nature, but this is not to say that significant amounts of processing effort can't still be expended. For example, in the case of key-exchange,

the transactions are very limited in both scope and duration and although the time available to arrive at a solution is still measured in seconds, as opposed to weeks or even minutes, this still permits a great deal of processing burden compared to a system that must be responsive on the microsecond or millisecond timescale. Other systems are likely to involve mobile units, perhaps including small, unmanned aerial vehicles, where space, weight, and power requirements drive the design. Still others might involve distributed sensor networks made up of thousands, perhaps millions, of remote sensors that have to be extremely low cost, and probably small and low powered, to be viable. However, the asymmetry of the responsibilities between sender and receiver might permit the receiving station to carry a load burden that the sender could not accommodate. Finally, in addition to being scalable with respect to cost, the solution must also be scalable with message length. In order to be suitable for key exchange purposes, it will be necessary to transfer messages that are on the order of several hundred to perhaps a few thousand bits. Algorithms that are not of polynomial time or better are categorically unsuitable. In fact, the limitations of algorithms that are not near-linear or better in complexity are likely to prove too severe for realistic message lengths.

6.3 Concurrent Codecs in the OSI Network Model

While not of strong relevance to the work here, some may find it useful to envision where a concurrent codec fits within the traditional framework of the OSI seven-layer network model. In theory, the jam resistant behavior afforded by a concurrent codec can be implemented via the interaction of elements operating at several layers and, as with most network implementations, the actual partitioning of tasks is not rigidly defined and so some implementations will allocate some tasks to different layers than

other implementations. Accordingly, the discussion here presents merely one possible partitioning.

Ideally, all elements of the codec would be implemented in the physical and data transport layers. The physical layer, being responsible for the actual transmission and reception of the data packets, dictates the channel error properties and how asymmetric they are. Therefore it plays a major role in the physical realization of the type of jam-resistant channels under consideration and cannot be ignored. The data link layer is responsible for ensuring data integrity between nodes (as opposed to end-to-end data integrity) and since our purpose is to permit continued communications between a sender and a receiver across a single wireless link, we are intrinsically discussing a node-to-node protocol. Therefore it is most appropriate that the bulk of the codec, the portion that extracts the list of legitimate codewords and/or their corresponding messages from the packet, be implemented in this layer.

However, an integral part of the overall protocol is the discrimination of false messages. While this should ideally be done at the data-link layer, it might be more convenient to pass the entire extracted message list up to a higher layer. That determination will primarily be driven by how false messages are discriminated against. For example, each message could contain a digital signature that could be used to screen out illegitimate messages.

6.4 Relevant Performance Metrics

To evaluate codec designs, it is necessary to devise suitable measures for their performance against the three often competing requirements of jam resistance, efficiency, and scalability. Unfortunately, as previously discussed, the truly relevant metrics cannot be developed in isolation to the intended application. We can, however, develop a set of abstract metrics that are useful in comparing potential codec designs relative to

each other. Since jam resistance is the primary requirement, a quantitative metric for that measure will be used. For the other two, we will simply note that the efficiency and scalability are generally related to the amount and type of processing that must be performed and also to the degree to which that processing can be parallelized.

Any suitable metric for jam resistance must relate the amount of effort expended by the attacker to the detrimental effect that effort has on the receiver. In traditional systems, this generally takes the form of determining the effect that the jammer's average transmission power has on the receiver's bit error rate (BER). While this metric will be examined when physical layer performance is considered in another section, it is not sufficient by itself because of the highly asymmetric channels employed and because there is considerable tolerance to space errors. In addition, most systems that employ error correcting codes do not take into account the computational effort associated with that coding (at least not when discussing generic performance expectations). This is because it is generally assumed that errors, even if caused by hostile jamming, are randomly distributed within a codeword. Unfortunately, a concurrent codec operates under far less favorable conditions since an adversary can specifically seek to overwhelm the decoder's ability to extract messages from the bit stream. In fact, in many systems the amount of additional work the receiver can handle will determine the performance limit of the system and not the point at which the BER rises precipitously. As a result, we also need a metric that relates the amount of effort expended by the attacker to the computational burden placed on the decoder.

The attack effort is most naturally defined in relative terms: the ratio of the energy broadcast by the attacker to the energy broadcast by the message originator. As will be shown in a later section, most feasible modulation schemes will have to be some form of a generalized On-Off Keyed (OOK) system in order to produce the necessary channel asymmetry. Thus, for a generic treatment, we will assume that each mark that is broadcast equates to a certain amount of energy expended and that spaces

require no energy expenditure. Thus the ratio of energies, which will also equal the ratio of average powers, is simply the ratio of the packet mark densities

$$\zeta_A \triangleq \frac{\mu_A}{\mu_S} \tag{6.2}$$

where μ_A is the mark density of the attacker and μ_S is the mark density of the legitimate sender. We will generally express this ratio in decibels, which is reasonable since it is a measure of relative power.

We should note that this ignores several issues. For instance, if the jammer is nearby, they might be able to create a mark in the packet using considerably less energy than the legitimate sender. Conversely, if they are further away they may have to use considerably more energy to do so. However, this shortcoming is the same for the analysis of traditional systems as well and the normal approach is to discuss the problem in terms of the signal at the receiver's input leaving it to the person solving a particular problem to convert that into the actual power ratios between the transmitters. That same concept applies here.

Measuring the receiver's effort also has a natural definition, although it is somewhat more arbitrary. A concurrent codec must recover a list of all possible messages that are covered by the received packet. The receiver must then process those messages to discriminate which, if any, are genuine. The channel becomes jammed if either the concurrent codec requires more time or resources than are available to generate the message list or if the list is too long for the receiver to process within its time and resource budget. In most cases, the processing effort required for both tasks, recovery and discrimination, scales with the length of the message list. Therefore, the ratio of the number of messages recovered, M_R, to the number of messages that would have been recovered in the absence of jamming, $M_R^{'}$, will be used as the

primary metric

$$\zeta_R \triangleq \frac{M_R}{M_R'} = \frac{M_R}{M_S} \tag{6.3}$$

In most cases, the number of messages that would be recovered in the absence of jamming is simply the number of genuine messages sent; when this is not the case it will be dealt with appropriately. Furthermore, specific algorithms may have more suitable ways to measure relative decoder workload, which will be used where appropriate. Like the attacker's effort, the receiver's effort will also usually be expressed in decibels, even though the relationship to power ratios is only inferred. Additional appropriate metrics will be defined, if needed.

If we plot the relative receiver effort versus the relative attack energy we obtain a performance characteristic for that concurrent codec. As mentioned previously, both the receiver and the attacker have associated limits - the receiver has a maximum amount of effort they can afford to expend to recover the message while the attacker has a maximum amount of energy they can commit to the attack. The intersection of these two limits serves as the performance criterion for the system and determines the acceptability (for jam resistance) of the codec. If the criterion lies above the characteristic, then the receiver can process any amount of jamming the attacker is willing to generate. Conversely, if the criterion lies below the curve, then there is an amount of jamming the attacker is willing to produce that will overwhelm the receiver's ability to cope.

Quantitatively, it is impossible to place exact, or even static, limits on either the amount of relative effort the receiver can afford to expend or the amount of energy the attacker is willing to commit. In fact, concurrent codecs that have fixed performance curves are probably not desirable. Instead, we want a codec that is extensible so that, by changing a few parameters, the curve can be shifted to a more advantageous location - usually in exchange for more absolute effort on the part of both the originator and the recipient. This is analogous to using a cryptographic

system that permits the users to select longer length keys to gain cryptographic security in exchange for longer encryption and decryption efforts.

We can, however, gain some appreciation for the performance graphs by noting that if a receiver has to process ten thousand messages to find the one original message, it is expending an additional 40 dB of effort. This is a processing burden likely to be well within the capabilities of even moderately capable machines for many applications. Conversely, if the attacker is transmitting a hundred times as many marks as the originator, then they are expending 20 dB of relative energy. Even if they can afford the energy and the equipment burden to operate at this level, such transmissions are likely to attract unwanted attention.

6.5 Inadequacy of Conventional Coding Theory

We now turn our attention to actually constructing a suitable concurrent code. At first glance, it is tempting to consider the use of conventional, or even specially developed, error correcting codes to impart the necessary jam resistance to an insecure channel. Certainly conventional coding theory, via the use of error correcting or error correcting codes to mitigate non-malicious channel noise, is a well established and developed field that has been extensively applied to mitigating the effects of jamming in channels where the security of the key remains intact. This success can lead to the misperception that error correcting codes can be used to combat jamming in general. A subtle point, however, is that an intact key robs the attacker of the information needed to create a devastating low-power attack signal; in essence, the key turns any attack signal into essentially random noise and error correcting codes are very good at dealing with random noise.

However, error correcting codes cannot function well in the presence of noise that has been carefully crafted by a hostile party who possesses all of the information that

the genuine sender does (i.e., has obtained the key). To see that this is the case, one need only consider the basic goal of an error correcting code and its subsequent limitations. That goal is to recover the originally transmitted codeword from a data stream that has undergone some form of corruption. This, in turn, is done by examining the received packet and determining which codeword is the most likely one to have been transmitted. But if the attacker simply transmits a second perfectly valid codeword, the receiver will be left with a packet that is some combination of the two (regardless of what type of communications channel is used). Even if the receiving codec could determine that two codewords were combined (something that would be highly unlikely), it would still have to identify which two and, even then, would be unable to determine which codeword was genuine and which was the rogue.

The bottom line is that error correcting codes are designed to recover a single codeword corrupted by essentially random noise, but an attacker that corrupts the original codeword by transmitting additional legitimate codewords on top of it is employing noise that is anything but random.

6.6 Performance of a Straight-through Codec

In order to motivate thinking about concurrent codes in terms of why they may or may not work well, we begin with a straight-through concurrent codec, which performs no encoding - the transmitted codeword and packet are identical to the original message. Not surprisingly, this does not work well. With no encoding, even in the absence of any noise (malicious or otherwise), the decoder will generate a list of 2^d potential messages where d is the degree[1] of the packet. Thus codewords of degree one or greater are intrinsically *autohallucinogenic* (i.e., they cover other codewords even without corruption).

[1]The degree of the packet is the number of marks in it. Also known as the Hamming weight.

Even worse, every additional mark inserted by an attacker is highly hallucinogenic since it further doubles the length of the message list. Hence an attacker can expend far less energy than the sender while forcing the receiver to do exponential work. The performance characteristic for this codec is

$$\zeta_R = 2^{(\mu_S(1-\mu_S))(m\zeta_A)} \ (\zeta_R \leq 2^L) \tag{6.4}$$

To emphasize how inadequate this codec is, for a performance criterion of 20 dB of attack effort and 40 dB of receiver effort, a message with 50% marks would have to be shorter than half a bit in length!

Another point to keep in mind is that our definition of receiver effort specifically excludes the baseline effort required in the presence of no jamming signal and that effort is also exponential in terms of the degree of the legitimate message. It's not too much of an exaggeration to proclaim that the genuine sender is the receiver's worst enemy.

6.7 Performance of a One-Hot Codec

The obvious flaw of the straight-through concurrent codec is that legitimate codewords are autohallucinogenic. To overcome this, we need to construct a codebook in which no codeword is a subset of any other codeword - or, at least, that this is a vanishingly rare occurrence.

Ideally, our codewords would be orthogonal to each other such that a given mark can never be used as part of two distinct codewords. If this is the case, then an attacker must exert as much energy as the sender to inject a rogue message and the length of the list grows only in direct proportion to the amount of energy expended - it is impossible to generate hallucinations. Thus the attacker cannot force the receiver to do a disproportionate amount of work. However, this also establishes a best-case

performance bound since the receiver can always force the receiver to perform work in direct proportion to the amount of energy expended.

Such a codebook is actually trivial to construct. The simplest concurrent codec that exhibits this property is a one-hot concurrent codec in which an L-bit message is reduced to a single bit located in a unique position within a 2^L-bit codeword. The originator can therefore send M_S messages concurrently by asserting the associated M_S marks in the packet. As before, we assume that the energy required is proportional to the number of marks which, in this case, is also proportional to the number of messages. The resulting performance characteristic is therefore

$$\zeta_R = \frac{M_R}{M_S} = 1 + \left(1 - \frac{M_S}{2^L}\right)\zeta_A \tag{6.5}$$

As expected, the attacker can only force the receiver to perform linear effort. It might be tempting to note that, since M_S will almost certainly be small compared to the number of bits in the packet, the attacker will have to expend nearly as much energy as the receiver. However, it must be remembered that our definitions of "energy" for the two are quite different, hence at the very least there is a proportionality constant involved.

A one-hot codec would appear to be an ideal solution to the jam-resistance problem. In fact, it marks the best possible performance of any conceivable concurrent codec in an OR-channel facing optimal hostile jamming. The problem is scalability.

To send a message consisting of L bits, the packet must contain 2^L bits. In a hypothetical system capable of transmitting packets at a rate of one gigabit per second, a sixty-four bit message would require over half a millennium to transmit. Hence the exponential scaling of the packet length makes this approach unworkable.

6.8 Performance of a Superimposed Codec

The one-hot concurrent codec attains the ultimate level of jam resistance because every codeword is orthogonal to every other codeword. However, to achieve this degree of jam resistance, both the sparseness of the codebook and the sparseness of each codeword is at the ultimate level of one mark per codeword. Recognizing this, plus the fact that we do not need the ultimate performance, only performance that is good enough, we can trade some jam resistance for scalability improvements.

Consider a codec that maps L-bit messages onto C-bit codewords of degree d, meaning that each message is transformed into a codeword containing d marks. This is also referred to as a C-bit codeword of "Hamming weight" d. The number of actual codewords, C_A, is simply the total number of possible messages, namely 2^L, while the total number of potential codewords, C_P, (codewords that fit the required pattern, namely C-bit codewords of Hamming weight d) is

$$\begin{aligned} C_P &= \left(\begin{array}{c} C \\ d \end{array} \right) \\ &= \frac{C!}{d!(C-d)!} \end{aligned} \tag{6.6}$$

The codewords in such a system cannot be orthogonal since, by the pigeon-hole principle, a mark at a given location is necessarily going to be a component of many different codewords. So we need a different measure for how "different" the codewords are from each other. A common measure is the "Hamming distance", which is the number of places in which two codewords differ. It is equal to the Hamming weight of the bitwise-XOR of the two codewords. Strictly speaking, since it is only the marks that count, a better metric would be the number of marks in one codeword that are not in the other. This would recognize that fact that codeword A need not be the same distance from B as B is from A. As a trivial example of this distinction, imagine

that the codeword A is covered by codeword B, meaning that all of the marks in A are already in B, but that B has a few additional marks. In that case, some of the spaces in A would have to be converted to marks in order to convert A into B. But nothing has to be done to B since it already encompasses A (in fact, it is autohallucinogenic) and decoding B will also produce A. However, if every codeword in the codebook has the same Hamming weight, then this distinction becomes irrelevant because for every mark that is in A that is not in B, there must also be a mark that is in B but not in A. If the Hamming weights are not all equal, then this is an issue that becomes relevant as the variation in Hamming weight across the codebook grows. Except where needed, we will assume that the variation is practical codebooks is sufficiently small as to justify ignoring this distinction and using the normal Hamming distance as our metric of codeword separation.

If the expansion factor, F, (i.e, ratio of codeword length to message length) is even moderately high, then the codebook becomes extremely sparse. If, in addition, the codeword degree is relatively small then each codeword is also very sparse. The combination of these two effects can be used to create large Hamming distances between codewords.

Two approaches to constructing the codebook are possible: (1) devise an algorithm specifically designed to produce optimally spaced codewords codewords, or (2) use a pseudorandom process to generate codewords and determine if codec parameters (expansion and degree) can be found that make the codewords far enough apart to be sufficient for practical purposes. The first, while attractive, may be difficult to achieve, especially since the packets must also be efficiently decodable. There is no guarantee that the second approach will work or that the resulting codewords will be short enough to be practical, but it leaves the avenue open for first developing an efficient means of encoding and decoding packets and then determining if the resulting codebook is sufficiently jam resistant.

In developing the performance characteristic for this codec, we must consider the best way to construct attack packets. The attacker has three basic options: (1) they can transmit random marks, (2) they can construct packets by combining legitimate codewords, or (3) they can construct packets specifically designed to produce a disproportionately large number of hallucinations with a minimal number of marks. The first option is clearly the simplest to implement, but assuming that the attacker will do so should a better option exist is not realistic, especially since the entire motivation for developing concurrent codecs is because we have assumed that an attacker that obtains the key for a traditional jam-resistant system is willing to put forth the effort to exploit it. The second option at least permits the attacker to inflict a baseline level of harm on the receiver since they can force the receiver to recover and process the messages they've included. The third option is clearly preferred, but one of the goals of codebook design will obviously be to either make such packets nonexistent or make them extremely hard to find. Making them nonexistent is clearly achievable in the limit - a one-hot codec achieves this. Making ones that do exist extremely hard to find is achievable by using cryptographically sound one-way hash functions in the generation of the codewords. Thus, for now, we will assume that this option has been rendered infeasible.

Beginning with our working definitions of attacker and receiver effort and assuming (1) that an adequate pseudorandom function is used to generate the codewords and (2) that the attacker uses the second method to construct their attack packets, the

performance characteristic is given by the following somewhat horrific equation[2]

$$\begin{aligned}\zeta_R &= 1+\left(\frac{1}{M_S}\right)\left(\frac{\eta}{L}\right)\frac{\ln\left(1-\zeta_A\left[1-\left(1-\frac{1}{LF}\right)^{\frac{M_S L}{\eta}}\right]\right)}{\ln\left(1-\frac{1}{LF}\right)}+ \\ &\left(\frac{1}{M_S}\right)\left(2\left(\zeta_A\left[1-\left(1-\frac{1}{LF}\right)^{\frac{M_S L}{\eta}}\right]\right)^{\frac{1}{\eta}}\right)^{L}\end{aligned} \tag{6.7}$$

The performance characteristic (Figure 6.8) is very insensitive to the message length or the mark factor while the expansion factor, F, sets an upper limit on the performance. Because the codewords become orthogonal in the limit of infinite codeword factor ($F \gg s$), the superimposed concurrent codec's characteristic for infinite codeword factor is identical to that of the one-hot concurrent codec, which was shown to represent the theoretical performance limit. For finite codeword factors, the attacker is constrained to the characteristic for the one-hot codec until they have committed an amount of energy that causes hallucinations to be produced in significant numbers. However, once this break-over energy level is reached, the hallucinations grow exponentially creating a near vertical wall effectively jamming the channel immediately.

[2]See Appendix F for the derivation of 6.7

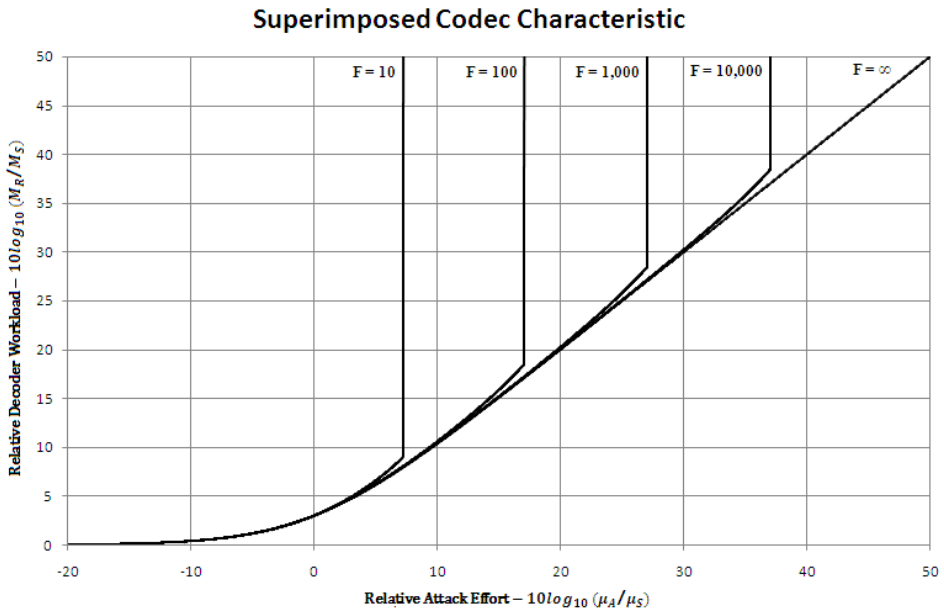

Figure 6.1: Performance characteristic of a superimposed concurrent codec

Chapter 7

BBC-based Concurrent Codecs

7.1 Standard BBC Algorithm

Although the superimposed codec performs acceptably from a jam-resistance viewpoint, its practicality is dependent upon efficient encoding of codewords and, more importantly, decoding of packets. As the literature search suggests, no such efficient algorithm has existed up to this time. This most likely owes more to the lack of a driving need for one rather than to its intrinsic difficulty. None-the-less, what is presented here is believed to be the only codec algorithm whose performance is of linear complexity in terms of both the message length and the number of messages (codewords) in a packet. The name "BBC" is derived from the last initials of the three co-developers of the algorithm, namely Leemon Baird, William Bahn, and Michael Collins. The "Standard" qualifier denotes that it is the most basic variant of the BBC algorithm; several enhancements have been developed to address practical considerations. These enhancements are the subject of a later section.

The approach taken by BBC to efficient decoding is to construct an algorithm that permits decoding arbitrary codewords in a fashion nearly identical to how packets are tested for specific codewords in most superimposed coding schemes. That approach

simply looks to see if each mark in the codeword is present in the packet. As soon as a needed mark is not found, the answer is known. On the other hand, if the search progresses to the end of the codeword without finding a missing mark, then the codeword is covered by the packet. What has prevented efficient decoding to date is that most superimposed coding schemes use codewords that are constructed based on the entire message all at once. For instance, Bloom filters generate codewords by running the message through d different hash functions to produce the d necessary mark positions. However, if we construct codewords from the message one bit at a time, we can decode the packet the same way. More precisely, we must construct codewords in such a way that progressively larger subsets of codeword marks can be determined using progressively larger subsets of message bits. Implied in this description is that each subset must contain all prior subsets.

7.1.1 Core Coding/Decoding Algorithm

The simplest and most obvious way to build a set of progressively larger subsets of the message to be encoded is to use progressively longer prefixes of that message. For example, if the 4-bit message 0110 is to be encoded, the series of prefixes that would be used would be $\{0, 01, 011, 0110\}$. Each prefix would be used to generate one or more mark positions. This is easily done using an appropriate hash function. To be appropriate, the hash function must be able to accept inputs of arbitrary length and must generate outputs that cover at least the range of the number of bits in the codeword. If the output spans a larger range, it must be mapped onto the range. In addition, the hash function should be reasonably random, even after any necessary range mapping.

Thus a codeword is generated using the following algorithm:

```
ENCODE(L, H, M) =
    for i = 1 to L
        Place a mark in the codeword at position H(M[1:i])
```

where $M[1:i]$ is the first i-bits of the message and $H(x)$ is a hash function that outputs a value from 1 to C (the number of bits in the codeword).

The decoding can be done using a recursive depth first search starting with $i = 0$:

```
DECODE(L, H, M, i) =
    if (i equals L)
        Add M to the message list
    else
        if there is a mark at H({M[i:1],'0'})
            DECODE(m, H, {M[1:i],'0'}, (i+1))
        if there is a mark at H({M[i:1],1})
            DECODE(m, H, {M[1:i],'1'}, (i+1))
```

where the notation $\{M[1:i], `0'\}$ means to concatenate the string $M[1:i]$ and the string '0', in other words, add a 0 to the end of the partial message string. Note that M[a:b] is taken to be an empty string if $b < a$.

The decoding algorithm is as follows: After decoding the partial message consisting of the first i bits of a message, determine where the encoder would have placed a mark if the next message bit were a zero. If a mark is found at that location, add a zero to the partial message and proceed to decode the next bit. Repeat this process for the case of the next message bit being a one. The decoding efficiency is obtained by the fact that once a prefix is found that is not covered by the packet, no other messages sharing that same prefix will be considered.

The following examples show how the encoding algorithm works. The hash function for these examples is shown in Table 7.1.1. In this table, the columns labeled w to z contain the 1-bit through 4-bit message prefixes, respectively, while the columns $W(w)$ through $H(z)$ contain the hashes of the corresponding prefix. The outer two columns are simply the message identifier. Future hash tables of this sort will exclude the message prefix columns, as these are readily discernable from the structure of the table itself.

```
Example 1 - message: 0110

    Prefix #1:    0    H(0) = 12
    Prefix #2:   01   H(01) = 15
    Prefix #3:  011  H(011) = 20
    Prefix #4: 0110 H(0110) = 25

    Codeword: 00000000 00010010 00010000 10000000

Example 2 - message: 1110

    Prefix #1:    1    H(1) = 10
    Prefix #2:   11   H(11) = 14
    Prefix #3:  111  H(110) =  1
    Prefix #4: 1110 H(1110) = 21

    Codeword: 10000000 01000100 00001000 00000000

Example 3 - message: 0010

    Prefix #1:    0    H(0) = 12
    Prefix #2:   00   H(00) = 19
    Prefix #3:  001  H(001) = 23
    Prefix #4: 0010 H(0010) = 19

    Codeword: 00000000 00010000 00100010 00000000
```

Example #1 and #2 demonstrates that just because two messages are nearly identical does not indicate that their codewords will be related at all. Although these two messages differ only in the first bit, the resulting codewords are completely disjoint. However, two codewords that share the first n-bits will have at least the marks resulting from the first n prefixes in common. After the first prefix in which two messages differ, the remaining codeword marks for those two messages are independent and unrelated (assuming a perfect hash function). None-the-less, every codeword has at least one other codeword that differs from it by at most one mark. A similar loss of Hamming distance in a conventional error-correcting code would almost entirely

Table 7.1: Toy Hash Table

#	w	H(w)	x	H(x)	y	H(y)	z	H(z)	#
0	0	12	00	19	000	6	0000	6	0
1							0001	15	1
2					001	23	0010	19	2
3							0011	14	3
4			01	15	010	10	0100	7	4
5							0101	5	5
6					011	20	0110	25	6
7							0111	2	7
8	1	10	10	28	100	11	1000	13	8
9							1001	3	9
10					101	8	1010	30	10
11							1011	29	11
12			11	14	110	28	1100	12	12
13							1101	17	13
14					111	1	1110	21	14
15							1111	9	15

defeat the purpose, whereas in a concurrent code, it merely results in additional processing effort on the decoder's part to recover the message. In essence, a situation in which a small amount of noise in a conventional system could be catastrophic results in only a small amount of additional effort in a concurrent code. None-the-less, we will see how the nominal Hamming distance of the codebook can be increased to reduce the decoder's processing burden.

Example #3 shows a different artifact, namely that it is not guaranteed that all of the marks generated for a given codeword will be disjoint. In this case, two different prefixes of the same message produced the same mark position resulting in a codeword of containing only three marks instead of the nominal four. Worse, since the redundant mark position also happens to be the last mark position, the codeword for message 0011 will also contain all of the marks for message 0010 and hence be autohallucinogenic; we will shortly see how to make encountering such a codeword exponentially unlikely.

Turning our attention to the decoding algorithm, the following examples also use the hash function from Table 7.1.1.

```
Example 4 - Codeword: 00000000 00010010 00010000 10000000

    H(0) = 12 ? Y
        H(00) = 19 ? N
        H(01) = 15 ? Y
            H(010) = 10 ? N
            H(011) = 20 ? Y
                H(0110) = 25 ? Y (Message: 0110)
                H(0111) =  2 ? N
    H(1) = 10 ? N

    Message list: {0110}

Example 5 - Codeword: 00000000 01111100 00100010 00010000

    H(0) = 12 ? Y
        H(00) = 19 ? N
            H(000) =  6 ? N
            H(001) = 23 ? Y
                H(0010) = 19 ? Y (Message: 0010)
                H(0011) = 14 ? Y (Message: 0011)
        H(01) = 15 ? N
    H(1) = 10 ? Y
        H(10) = 28 ? Y
            H(100) = 11 ? Y
                H(1000) = 13 ? Y (Message: 1000)
                H(1001) =  3 ? N
            H(101) =  8 ? N
        H(11) = 14 ? Y
            H(110) = 28 ? Y
                H(1100) = 12 ? Y (Message: 1100)
                H(1101) = 17 ? N
            H(111) =  1 ? N

    Message list: {0010, 0011, 1000, 1100}
```

Example #4 shows the basic decoding tree for a single codeword with no complications. Example #5, on the other hand, shows the decoding tree for a packet

that was generated by combining the codewords for messages 1000 and 0011. It has already been noted that 0011 is autohallucinogenic, so we expect 0010 to also appear in the final message list. In addition, because of "*mark collisions*", the marks in the two codewords combine to cover yet a fourth message, 1100. False messages, like this, that remain after decoding the message bits are called "*terminal hallucinations*" while those that exist at prior decoding stages are referred to as "*working hallucinations*".

While not present in this example, it is easy to see that an attacker could easily produce many more messages with only a handful of inserted marks by simply "fleshing out" the end of the decoding tree to cause it to produce messages "close to" ones that are already there. Such a tree is called "*bushy-tailed*"

To a large degree, the issues shown by these examples disappear for messages and codewords of practical interest, namely messages that are hundreds of bits long and codewords that are tens of thousands, or even millions, of bits long (keep in mind, this large expansion factor represents not only the code rate, but also the spectrum spreading factor). The issue of small Hamming distances for lots of small groups of codewords still remains, but that will be remedied shortly.

At this point, we can list several properties of the primitive BBC codebook that at least have the potential of causing problems. We can then discuss each and perform any necessary analysis and/or mitigation. A BBC codebook, as we've described it thus far, has:

1. non-orthogonal codewords.
2. codewords with small Hamming distances.
3. mark collisions.
4. terminal hallucinations.
5. vulnerability to bushy-tailed attacks.

Let's take each point, at least briefly, in turn. More detailed discussions will be presented in the following sections.

(1) The fact that the codebook is non-orthogonal is not unique to BBC; practical error correcting codes are similarly non-orthogonal.

(2) The fact that the minimum distance in a primitive BBC codebook is, at most, one (and, in practice, is almost certainly zero) is not a significant problem. This is a significant distinction relative to traditional error correcting codes in which the ability of the code to correct errors is dictated by the minimum Hamming distance of the codebook and in which a minimum Hamming distance of zero would be intolerable. But in a concurrent code, the correct message will still be received, but it will have to be identified from a potentially large number of hallucinations that are also in the final message list. Crafting a message format that makes this doable is fairly straight-forward (a simple traditional checksum would likely suffice), but simply adding terminal checksum bits to the message virtually eliminates this issue.

(3) While the location of marks placed into a codeword are expected to collide with marks in other codewords (or even prior marks in the same codeword) with some regularity, this does not equate to a hash collision in the usual sense of the term. The mark location is merely derived from the internal state of the hash function, which is generally much larger.

(4) Terminal hallucinations that appear coincidentally can be exterminated exponentially by terminal checksum bits. In the rare case that any hallucinations survive the entire decoding process and become "*realized hallucinations*", they can be filtered out by the messaging protocol, such as the inclusion of traditional checksums or digital signatures.

(5) Bushy-tailed attacks are a potentially serious problem but can be effectively mitigated by prepending messages with a random preamble or a message hash and constructing the authentication rules to take advantage of them.

7.1.2 Use of Checksum Bits to Increase Hamming Distance

The low minimum Hamming distance for the primitive BBC codebook describe previously is a simple consequence of differing in only the last bit. As prior examples demonstrate, two messages can differ by only one bit and still be be completely disjoint in the location of their marks provided that bit in which they differ is the first bit. In general, the more message bits that remain after the longest common prefix shared by two messages messages, the greater the Hamming distance will (probably) be between the resulting two codewords. Therefore, if we wish to increase the Hamming distance and reduce the number of hallucinations, we need to append additional bits that result in additional codeword marks such that the locations of those marks are a function of the entire message. In traditional coding, one way of doing this would be to compute and append a set of checksum bits to the message. In the BBC algorithm it is actually sufficient to simply append zero bits to the message because the hash value that results will automatically reflect the entire message as well as each additional bit in turn.

The streamlined hash table in Table 7.1.2 includes $K = 2$ checksum bits for each message.

The encoding algorithm is essentially the same except that the message is first modified by appending K checksum bits and then $(L + K)$ passes are made through the mark generation algorithm:

```
ENCODE(L, K, H, M) =
    M = {M,0(K)}
    for i = 1 to L+K
        Place a mark at position H(M[1:i])

Alternatively:
ENCODE(L, K, H, M) =
    for i = 1 to L
        Place a mark at position H(M[1:i])
    for i = 1 to K
        Place a mark at position H({M,0(i)})
```

Table 7.2: Toy Hash Table w/Checksum Bits

0	12	19	6	6	2	3	0
1				15	32	31	1
2			23	19	8	9	2
3				14	17	30	3
4		15	10	7	30	20	4
5				5	7	26	5
6			20	25	3	27	6
7				2	26	22	7
8	10	28	11	13	26	29	8
9				3	16	16	9
10			8	30	3	19	10
11				29	11	10	11
12		14	28	12	1	30	12
13				17	4	2	13
14			1	21	8	27	14
15				9	31	13	15

where *0(x)* represents a string of x zero bits.

The decoding algorithm is also nearly the same, except that it leverages the knowledge that, once all of the basic message bits have been recovered, only zeros are possible for legitimate messages:

```
DECODE(i, L, K, H, M) =
    if (i EQ L+K)
        Add M to output message list
    else
        if (i < L)
            if there is a mark at H({M[i:1],0})
                DECODE(i+1, L, K, H, {M[1:i],0})
            if there is a mark at H({M[i:1],1})
                DECODE(i+1, L, K, H, {M[1:i],1})
        else
            if there is a mark at H({M,0(i-L)})
                DECODE(i+1, L, K, H, {M,0(i-L)})
```

Repeating Example #5, in which the messages 1000 and 0011 were combined, we end up with the following decode tree:

```
Example 6 - Codeword: 00000000 01111100 10100010 01011100

    H(0) = 12 ? Y
        H(00) = 19 ? N
            H(000) =  6 ? N
            H(001) = 23 ? Y
                H(0010) = 19 ? Y
                    H(00100) = 8 ? N
                H(0011) = 14 ? Y
                    H(00110) = 17 ? Y
                        H(001100) = 30 ? Y (Message: 0011)
        H(01) = 15 ? N
    H(1) = 10 ? Y
        H(10) = 28 ? Y
            H(100) = 11 ? Y
                H(1000) = 13 ? Y
                    H(10000) = 26 ? Y
                        H(10000) = 30 ? Y (Message: 1000)
                H(1001) =  3 ? N
            H(101) =  8 ? N
        H(11) = 14 ? Y
            H(110) = 28 ? Y
                H(1100) = 12 ? Y
                    H(11000) = 1 ? N
                H(1101) = 17 ? Y
                    H(11010) = 4 ? N
            H(111) =  1 ? N

    Message list: {0011, 1000}
```

Notice that the addition of the checksum bits eliminated both hallucinations, including the autohallucination associated with 0011. Further note, however, that there are costs associated with the checksum bits beyond the additional complexity and necessary computation. For instance, the mark placed at location 17 as a result of encoding the checksum bits for 0011 forced the decoder to continue working potential message 1101 beyond the point where it originally expired and, in fact, required the action of its own checksum bits to prevent it from becoming an hallucination. However, for practical systems, the processing burden imposed by adding checksum

bits is generally a couple of percent which is more than gained back by the elimination of terminal hallucinations they result in.

7.1.3 Expected Codeword Density

As previously defined, the mark density of a codeword is the fraction of bits in the codeword that are marks. The density is primarily determined by the expansion factor, F (recall that $F = C/L$). For sparse codewords, a very good approximation is simply the total number of marks, N, placed in the codeword divided by the length of the codeword. For Standard BBC, the number of marks is the sum of the length of the message and the number of checksum bits, K, appended to it. Thus the nominal mark density of a codeword is

$$\begin{aligned} \mu_{NOM} &= \frac{N}{C} \\ &= \frac{L+K}{FL} \\ &= \frac{1}{F} + \frac{K}{FL} \end{aligned} \tag{7.1}$$

The nominal density is a reasonable approximation for codeword densities below about 10%, but as codewords become more dense, the growing likelihood that additional marks placed in it by the encoding process will collide with marks already there places downward pressure on the actual mark density. To determine the expected density of a codeword of any density, we define the $n(j)$ to be the expected number of marks in the packet after the addition of the j^{th} mark. The probability that the the additional mark will be distinct is equal to the fraction of spaces still existing in the codeword. Hence

$$\begin{aligned} n(j+1) &= n(j) + \frac{C - n(j)}{c} \\ &= n(K)\Big(\frac{C-1}{C}\Big) + 1 \end{aligned} \tag{7.2}$$

where $n(0) = 0$.

This recursive relation can be solved in closed form by noting that that

$$n(j) = \sum_{i=0}^{j-i} \left(\frac{C-1}{C}\right)^j \tag{7.3}$$

which yields

$$\begin{aligned} n(j) &= \frac{1 - \left(\frac{C-1}{C}\right)^j}{1 - \left(\frac{C-1}{C}\right)} \\ &= C\left[1 - \left(\frac{C-1}{C}\right)^j\right] \end{aligned} \tag{7.4}$$

Thus, the expected codeword density after adding all N marks is

$$\mu(N) = 1 - \left(1 - \frac{1}{C}\right)^N \tag{7.5}$$

While this is the exact solution (for the expected number of marks, not the actual number since adding marks is a stochastic process), the typical numbers involved invite significant roundoff errors if this equation is used directly. Fortunately, we can obtain a very good approximation that is easy to compute by noting that, for codewords of practical length, $(1/C) \ll 1$. We can therefore exploit the fact that $\ln(1 + x)$ is approximately equal to x for very small x, yielding

$$\begin{aligned} \mu(N) &\approxeq 1 - e^{-\mu_{NOM}} \\ &= 1 - e^{-\frac{N}{C}} \end{aligned} \tag{7.6}$$

which provides very good agreement for codewords as small as ten bits and is practically indistinguishable for codewords longer than one hundred bits.

7.1.4 Expected Packet Density

It is tempting to assume that the density of a packet formed by superimposing the codewords for M messages would be approximately $M\mu(N)$, at least for sparse packets. However, while this can be a useful estimate in many situations, it can severely overestimate the packet density under certain circumstances even for very sparse packets. The reason is that not all of the marks in BBC codewords are independent relative to other codewords because two messages sharing the same p-bit prefix will have at least p identical marks. Once two messages no longer share a prefix, any remaining marks will be independent.

To see how severe this underestimate can be, consider a case where 1000-bit messages having 30-bit checksums are encoded into million-bit codewords. For a single codeword, the nominal density is 0.1%. If 32 messages were superimposed in a packet, the nominal density would be estimated at 3.2%. However, it is possible that these messages were specifically chosen to produce a "bushy-tailed" decoding tree by only having messages differ in the last few bits. In this case, all 32 messages could share a common 995-bit prefix require only an additional 63 independent marks to encode the bare messages plus 960 more independent marks to encode the checksum bits. Thus the total number of independent marks is only slightly over two thousand yielding a packet density of only 0.2%, or barely twice the single codeword density.

A similar crowding out effect occurs at the beginning of even randomly chosen messages, although it's impact is not nearly as severe. The culprit here is that a b-bit prefix can only have 2^b possible values and, even if they are all used, only requires $2^b - 2$ independent marks to encode. Thus, if 256 messages were encoded into a packet, the first eight bits would nominally contribute 2,048 marks. However, even if every possible prefix were used, the most they could actually contribute would be 510.

If M messages are chosen randomly and their codewords superimposed, the expected number of independent marks that will be added to the packet can be determined by considering the binary tree representing the messages. At each level, i, of the tree there are 2^i nodes through which all M messages must pass. For randomly selected messages, each node will have an equal probability of being in the path of a message. Thus, on average, the number of nodes that are passed through at each level obey the same statistics as the number of distinct marks in a codeword of length 2^i having M independent marks placed into it. For M independent messages of length L, the number of independent marks is therefore given by

$$N(M, L) = \sum_{i=1}^{L} 2^i \left(1 - e^{-\frac{M}{2^i}}\right) \tag{7.7}$$

For $L < \log_2(M)$, this approaches 2^L very rapidly, while for $L > \log_2(M)$ is approaches M very rapidly, as shown in Figure 7.1.4. Notice that it is reasonable to assume that all messages are independent beyond codeword bit positions that are twice the number needed encode unique prefixes to each message. For example, if 256 random 100-bit messages are encoded, then since eight bits are sufficient to encode a unique prefix for each message and therefore all messages are expected to contribute independent marks to the packet after the first sixteen bits. If the curve in Figure 7.1.4 were truly symmetric about the center point, then the first sixteen bits would contribute, on average, half a bit per message. Thus, if the length of the messages is at least twice the base-2 logarithm of the number of messages, we can get a good estimate for the number of independent marks needed for randomly chosen messages by

$$N(M, L) \cong M(L - \log_2(M)) \tag{7.8}$$

when $L > 2\log_2(M)$.

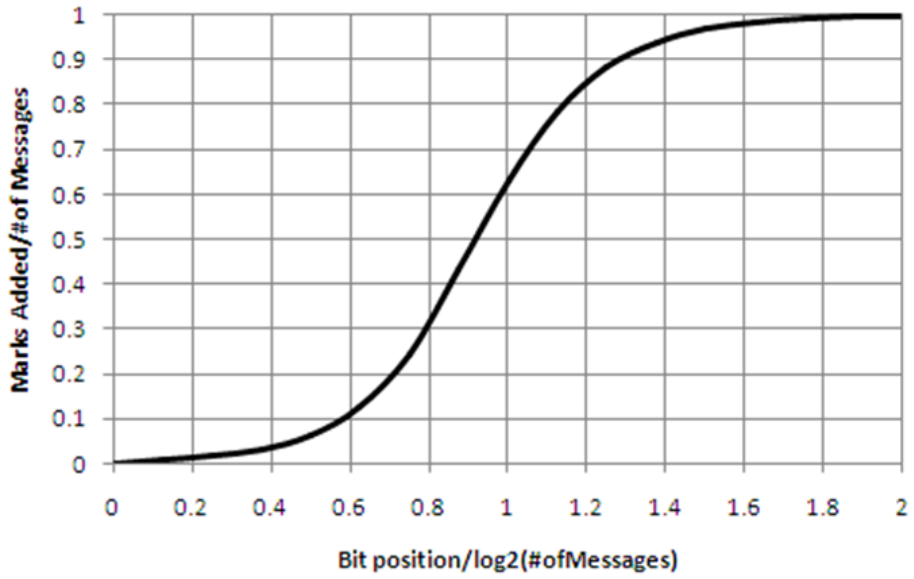

Figure 7.1: Independent marks associated with the leading bits of randomly chosen messages.

For the example described, namely 256 random 100-bit messages (no checksum bits), we would estimate a total of 23,552 independent marks while direct computation yields 23,834. Although this is a considerably closer estimate than the 25,600 marks in the case all marks being completely independent, even this estimate is more than adequate for most purposes. However, the point is clear that the number of messages encoded in a packet of a given density is a strong function of how that packet is constructed – a point not likely to be lost on an adversary interested in constructing good attack packets.

7.1.5 Hallucinations and Critical Density

A toy decoding tree is shown in Figure 7.1.5. Each filled node represents a node where a mark was found and that eventually led to a complete valid message. The unfilled nodes represent nodes where marks were found but which did not eventually lead to a complete valid message; each of these constitutes an hallucination that existed at that point. The dashed vertical line marks the transition between decoding information

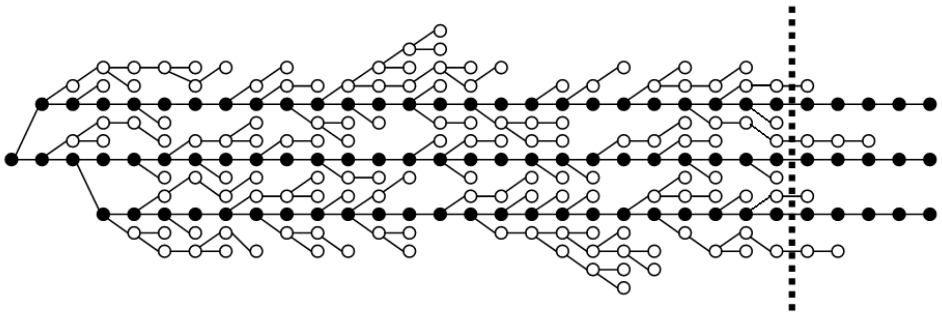

Figure 7.2: Typical decode tree for a toy decoder having 25-bit messages and 5 checksum bits.

bits and decoding the terminal checksum bits. As can be seen, no single hallucination survives very long, but as hallucinations die others are spawned. Since each node, filled or unfilled, in the tree requires the same amount of computation on the part of the decoder, we must ask what the hallucinogenic load is throughout the decoding process?

To answer this question, we can examine the behavior of the partial messages at an arbitrarily chosen point in the decoding tree. Each partial messages can be categorized as either belonging to a valid message, which is expected to survive the remainder of the decoding process, or to a hallucination, meaning that if it survives it will only be due to coincidental marks from other sources. Assuming that the decoding has progressed to a point where all valid messages have diverged, the hallucination generation possibilities are shown in Figure 7.1.5.

After decoding the n^{th}-bit of all of the messages, there are M actual messages and $H(n)$ hallucinations. For $n < L$, the next bit is an information bit (as opposed to a checksum bit) and therefore there are two possible paths to follow. For each actual message, one of these will correspond to the next bit in the actual message while the other corresponds to a position that, ideally, would not have a mark. If a mark is found at that location, then a new hallucination is spawned. The probability of this happening is simply the mark density of the packet, μ. For a hallucination, both

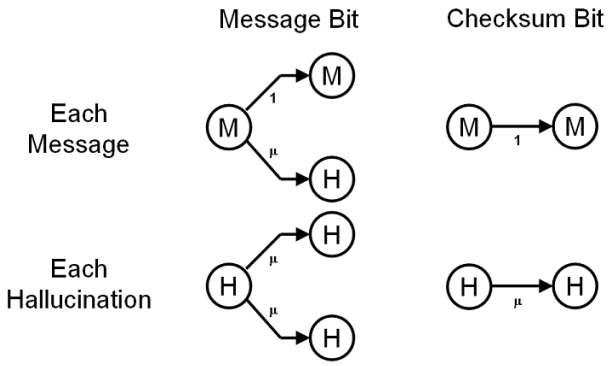

Figure 7.3: Hallucinations spawning paths during decoding

paths would ideally not have a mark and either path could result in a hallucination. Therefore, the expected number of hallucinations at the next step is

$$H(n+1) \cong \mu M + 2\mu H(n) \tag{7.9}$$

Assuming that the system achieves steady state, the expected number of hallucinations at one level of the decoding tree will be equal to the number at the next. Thus, the steady state hallucination rate, H_{SS}, is given by

$$H_{SS} = \frac{\mu}{1-2\mu} \tag{7.10}$$

and is the expected number of working hallucinations per valid message once steady state has been reached.

Examination of this relation shows that a packet mark density of 50% is expected to result in an infinite number of working hallucinations. In fact, 50% is the "*critical density*", μ_C, below which a steady state hallucination level exists and above which the number of hallucinations grows exponentially the further that processing continues (the number of hallucinations after decoding n bits goes as $(2\mu)^n$). Note that the total number of working hallucinations is directly proportional to the number of valid messages in the packet, be they from the genuine sender or an attacker. In

essence, each valid message acts as a seed for the generation of new hallucinations while existing hallucinations tend to expire exponentially. Somewhat surprisingly, the number of working hallucinations is quite tame until very near the critical density. For a density of 33% each valid message supports but a single working hallucination. Even as close to the critical density as 47.6% there are only ten working hallucinations per valid message. As a result, we can reasonably expect our codec to continue functioning until very close to the critical density and to fail very quickly at or above it.

While, on average, the burden due to working hallucinations is tame if the variance is high then it becomes possible for decoders to be overwhelmed too frequently to be practical. However, Fig 7.1.5 provides strong anecdotal evidence that the variance should not be too large since the actual number of hallucinations in steady state tracks very closely to the expected level throughout the decoding process. Furthermore, other simulations indicate that the variance of the total number of hallucinations during the decoding process is approximately equal to the number of valid messages. Thus, the standard deviation of the hallucination rate would be

$$\sigma_{HR} \cong \frac{1}{\sqrt{M}} \tag{7.11}$$

For this example, a thousand 200-bit messages were combined resulting in a packet density of 33.8%. Each message had just eight checksum bits appended to it. The smooth thin curve is the expected number of hallucinations at each stage of processing while the thick line is a plot of the actual number. The number of hallucinations is computed by counting the total number of message prefixes at each stage and subtracting the known number of actual messages. Because the message prefixes in the early stages are not long enough to even support the number of actual messages, the computed number of hallucinations is negative. The plot clearly shows that the processing burden has three distinct zones. There is a brief zone at the beginning of the

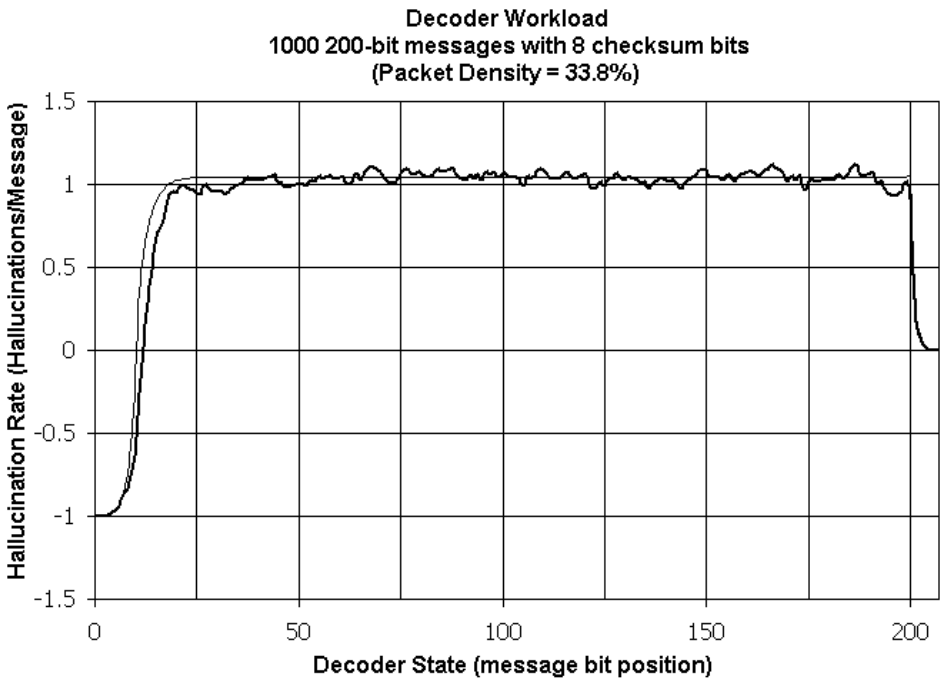

Figure 7.4: Comparison of actual and expected decoder workload.

tree in which the number of hallucinations grows exponentially. This then transitions into a long steady-state zone in which the number of hallucinations fluctuates closely around the steady state value. Finally, once the terminal checksum bits are reached, the number of hallucinations decreases exponentially.

7.1.6 Suppression of Terminal Hallucinations

As already shown, the addition of checksum bits increases the Hamming distance of the codebook and hence eliminates hallucinations that have otherwise survived the decoding process. Such hallucinations are called "*terminal hallucinations*". In designing a BBC-based concurrent codec, one parameter that must be chosen is the number of checksum bits to add. To make this determination, it is necessary to understand quantitatively how the number of checksum bits translates into protection from terminal hallucinations.

Assume we are working with a packet having a mark density μ. If those marks are randomly distributed (or if the mark location we test is randomly distributed), then the probability that we will find a mark when we test for one at a particular location is equal to the mark density. A terminal hallucination must survive K such tests, all of which are independent (given a suitable hash function), in order to make it to the final message list. As a result, the number of terminal hallucinations, H_T, that are expected to survive is given by

$$H_T = H_{SS}\mu^K \tag{7.12}$$

where H_{SS} is the number of steady state hallucinations prior to the checksum bits.

It is probably not excessive to consider adding a number of checksum bits that is on the order of 10% of the message length, so for messages that are, say, four hundred bits in length, forty checksum bits would be unlikely to result in an unacceptable increase in processing overhead. At a packet density of 50%, those forty checksum bits would translate into fewer than one in a trillion terminal hallucinations being added to the message list. If packet densities are not expected to exceed 33% and it is deemed adequate to allow one terminal hallucination in a million through to the final message list, then as few as thirteen checksum bits will suffice. It should be noted that the number of checksum bits needed for a given level of protection is dependent only on the packet density; in particular, it is independent of the message length or the expansion factor.

As anecdotal evidence of the checksum bit effectiveness, a packet was generated with four legitimate messages plus sufficient random noise to raise the packet density to approximately 53%. Since this is above the critical mark density (see next subsection), the decoding process was reduced to a crawl requiring over twelve hours to complete. The program used generates an ASCII log file permitting the reconstruction

of the decoding tree by printing out the message fragment each time a mark is found at a test location. The resulting file for this run was over 27 GB in size indicating approximately 250 million nodes in the decoding tree (in comparison, at 33% density there were approximately 2000 nodes). The theory predicts that approximately four million terminal hallucinations would result and this is in rough agreement with the number of hallucinations implied by the log file length. To eliminate this many hallucinations would require twenty-five checksum bits, so while it was reassuring, it was not surprising that the forty checksum bits actually used were adequate to eliminate all of them leaving only the four genuine messages in the final list.

However, as can be seen by the extreme case of the prior example, the number of final messages in the message list is not a good indicator of the workload burden experienced by the a BBC decoder. A much better metric is the number of nodes that are explored or, even better, the number of calls to the hash function. For each node in the decode tree that is visited, two calls to the hash function are made if the next bit is an information bit and one call is made if it is a checksum bit. Thus, for a single message with no false hits, a total of $2m + k$ calls to the hash function must be made. Each call is associated not only with the computation of the hash value itself, but also the memory accesses needed to check whether a mark exists at the resulting location. These two actions can be expected to constitute the bulk of the computational effort involved and even the bookkeeping associated with branching and backtracking scales tightly with the number of hash calls.

7.1.7 Typical parameters for a Standard BBC Codec

The most common codec configuration used in the software-defined radios used for field testing have been $\{2^{10}, 2^{20}, 2^5\}$ or, in other words, roughly thousand-bit messages, million-bit codewords, and 32-bit checksums. The mark density of a codeword for this configuration is only about 0.1%. Using such long, sparse codewords would

normally require either servoing the receiver to an embedded clock in the signal or using extremely accurate and stable oscillators at both ends. However, because of the inherent capabilities of concurrent codes, combined with the intrinsic packet-based decoding of BBC, neither is required and these issues can be accommodated, up to a point, with only a mild degradation in performance [76].

In analyzing the noise performance of a BBC-based system, it is important to note the distinction between a message bit and a mark. In particular, to transmit an L-bit message, the transmitter spreads its energy among N marks. For sparse codewords and a high ratio of message bits to checksum bits, equating the two is usually reasonable. However, in general, there is a concurrency efficiency, η, which is the number of message bits conveyed per transmitted mark. There are two factors that go into computing this efficiency. An approximation that is usually sufficient when considering a codec configured for an anti-jamming role is simply how many messages bits are represented by a codeword divided by the number of marks added to that message's codeword, or

$$\eta = \frac{L}{L+K} \tag{7.13}$$

For the typical codec parameters mentioned previously, the concurrency efficiency is 96.8%.

A closer approximation, which is usually relevant only when a codec is configured to relax jam-resistance in favor of data throughput, takes into account the fact that, except for the bookend marks, every mark added to the codeword by the encoder is placed at a location that is statistically independent of where every other mark is placed; hence some marks might coincide with a previous mark and not contribute any additional marks to the codeword. However, for the configurations of interest here, the effect is negligible.

As discussed in the early material, a concurrent codec combines the functionality of a traditional channel codec and spread spectrum modulator and, thus, blurs the

traditional distinction between a system's code rate and its processing gain. While rather arbitrary, we can consider the codec's concurrency efficiency to be the counterpart to a traditional channel codec's code rate. Where the correspondence works well is that, in a channel codec, the number of data symbols is increased the code rate, which the ratio of the number of bits input to the codec by the number of bits output by it, serves as a measure of how much the SNR-per-bit of the system is impacted by the channel coding. For instance, if the code rate is 1/2, then the effective SNR-per-bit is reduced by a factor of two (3dB) because the transmitter's total energy must now be spread over twice as many symbols. Likewise, in a concurrent codec, if the concurrency efficiency is 1/2, then the SNR-per-bit is reduced by a factor of two because each bit's energy allocation must be spread over twice as many marks. However, where the correspondence does not work well is when considering data rate. For a traditional system, a code rate of 1/2 either cuts the data rate, R, in half or increases the bandwidth requirements, W, by a factor of two. In either case, the R/W ratio is cut in half. In a concurrent codec, the concurrency ratio generally has no effect on either since the size of the codeword remains unchanged and the number of bits encoded into each codeword also remains unchanged. All the changes is the density of the codeword. Because there is not a reasonable across-the-board correspondence, we have therefore chosen to use a new and unfamiliar parameter, η, instead of using the traditional code rate parameter, since doing so would invariably lead to even more confusion and misunderstanding. Irrespective of what we name what, the important thing will be that we bring everything together correctly when we consider performance metrics, especially in comparison to traditional systems.

Before leaving this section, a few of the variants of BBC will be mentioned very briefly (see Section 7.2 for detailed discussions). One version, M-ary BBC, encodes multiple message bits with each mark. As in traditional M-ary systems, this allows more energy to be put into each transmitted symbol; however, a significant reduction

in the critical mark density accompanies this. Another variant inserts checksum bits within the message string, in addition to at the end. These can raise the critical mark density to arbitrarily high levels (short of 100%, of course), but lower the concurrency efficiency considerably. These two approaches can be combined, although no practical benefit of significance has been discovered.

A variant that could prove useful is multimark BBC, in which multiple marks are inserted for each message bit. The decoder then requires only that a specific minimum number of them, the quorum requirement, be present to declare the mark as being detected. This has the ability to make a limited number of mark errors (failure to detect a mark that is present) acceptable without losing the entire message. However, this lowers both the concurrency efficiency and the critical mark density, potentially making it better to use techniques such as erasure codes [9] across packets and/or traditional ARQ schemes [77] to recover lost packets.

7.2 Enhancements to the BBC Algorithm

Several enhancements to the Standard BBC algorithm have been developed, as well as other enhancements that are more directly related to the implementations of the codec and the preceding blocks than to the encoding/decoding algorithm itself. First, we will explore extensions of the basic encoding and decoding algorithm and then look at several enhancements to other parts of the signal processing chain.

7.2.1 Interstitial Checksum Bits

It is reasonable to ask if it is possible to increase the critical density by placing checksum bits within the message body in addition to placing them at the end [78]. For packet densities below the normal critical density of 50%, such interior checksum bits have virtually no utility because even if they reduce the working hallucinations

to zero at that point, the working hallucinations will rise back to the expected steady state level quite quickly. In fact, the addition of these bits is usually detrimental because they serve to increase the packet density, albeit marginally, and hence the number of working hallucinations will tend to increase.

But the situation becomes very different if packet densities are expected to exceed 50%. Here we can utilize interstitial checksum bits, which are simply checksum bits placed within the message, to clamp the exponential growth within acceptable limits. To do this, we decode a D-bit fragment of the message, during which the number of hallucinations grows exponentially; we then decode a string of S checksum bits that drive the number of hallucinations back down to a tolerable level. The resulting expected number of working hallucinations at the end of the message (and at the peak levels during message decoding) is then given by

$$H_{PK} = \mu \left[\frac{1 - (2\mu)^D}{1 - 2\mu} \right] \left[\frac{1}{1 - (2\mu)^D \mu^{(S+D)}} \right] \tag{7.14}$$

The factor $(1 - 2\mu)$ in the denominator of the first fraction, which in Standard BBC dictated the critical density, now divides into the numerator and is therefore a removable singularity. Therefore the "*extended critical density*" is governed by the denominator of the second fraction. The result is that the critical density is now

$$\mu_c = 0.5^{\left[\frac{D}{S+D}\right]} = 0.5^{\left[\frac{1}{1+\frac{S}{D}}\right]} \tag{7.15}$$

If one checksum bit is added prior to each message bit, then the extended critical density is 70.7% while if ten checksum bits are inserted prior to each message bit the extended critical density increases to 93.9%. In theory the extended critical density can be made arbitrarily close to unity, however at some point the fact that the numbers involved, such as number of hallucinations and the number of remaining spaces in the packet, are intrinsically discrete quantities comes in to play. However,

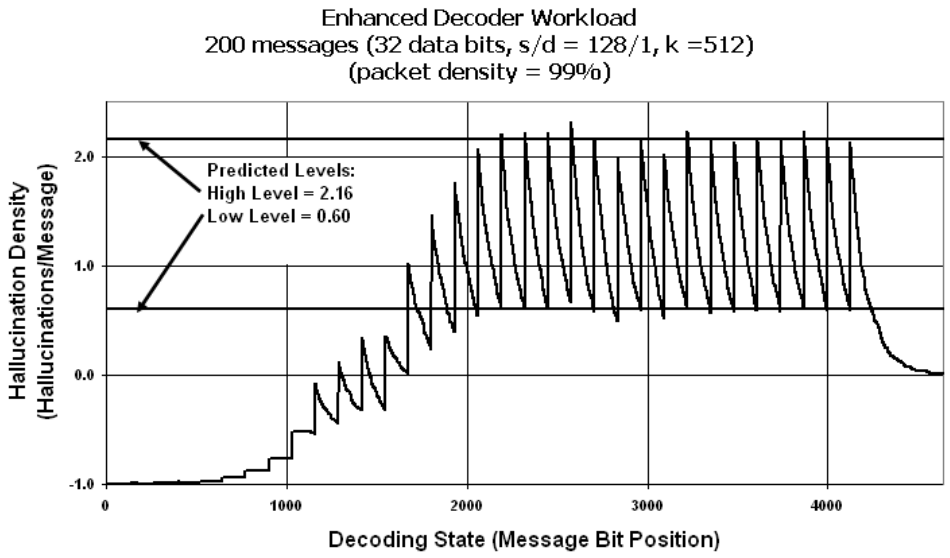

Figure 7.5: Decoder workload at extreme packet densities with interstitial checksum bits.

simulations have demonstrated that efficient decoding remains possible even with densities driven to 99.5% by prepending 128 checksum bits to each message bit and appending 512 checksum bits to the end. One such simulation is shown if Figure 7.2.1. The total number of calls to the hash function is only about 250 times the number of calls required for a Standard BBC decoder operating at a packet density of 33%. None-the-less, this is a significant penalty to pay for a capability that is of dubious practical utility.

It should be noted that being able to deal with packet densities above 50% has only limited utility in the case of an anti-jamming application. If the attacker is capable and willing to force the receiver's packet density to 50%, it is highly likely that they are capable and willing to force it another factor of two higher and completely jam the channel. Therefore the ability to extend the critical density is probably only of use when dealing with non-intelligent jamming sources (including environmental noise) and if that is the situation, then traditional jam-resistant approaches once again become a consideration, provided the necessary keys have been previously distributed or are publicly known. Hence about the only application where extended critical

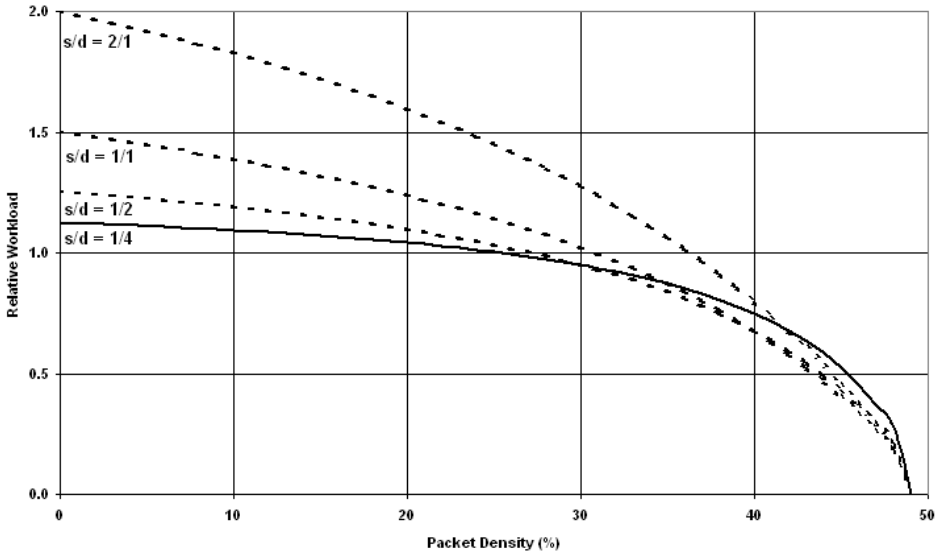

Figure 7.6: Number of hash function calls with interstitial checksum bits in comparison to Standard BBC.

densities might be useful, outside of academic circles, would involve initial exchange of keys in high noise environments without intelligent jamming.

Having said this, there is an additional benefit of interstitial checksum bits that make them potentially useful in a practical decoder. Figure 7.2.1 shows the workload of several codecs using interstitial checksum bits relative to the workload for a Standard BBC decoder as a function of packet density. The S/D ratio is the ratio of the number of interstitial checksum bits to the number of message data bits in each fragment of the padded message. Wherever the curve for a particular codec is below 1.0, the total number of hash function calls to decode a packet is less than required by a Standard BBC codec despite the longer message length. When operating at the threat model limit of 33% (the threat model is described in Section 8.1), several codec configurations result in a reduction of 10% to 15%. The 1/4 codec (represented by the solid line) would increase the critical density to 57% while increasing mark density in the codewords by approximately 25%. Whether this represents a reasonable tradeoff will depend on the specifics of a particular application, but in general is should be worth considering.

7.2.2 Multibit (M-ary) BBC

Since the fundamental signalling in a concurrent channel revolves around detecting the presence or absence of a mark and, more to the point, not in any modulation of the shape of that mark by the information being conveyed, a concurrent channel is fundamentally binary at this level. Unfortunately, this robs it of the ability to increase the bandwidth efficiency by using more than two symbols in its alphabet. However, the nature of the BBC algorithms lend themselves to encoding more than one bit into each mark simply by increasing the number of bits that are consumed by each call to the hash function. For instance, if each time the message prefix was extended two more bits of the message were incorporated, then there would be four possibilities for the new prefix and, therefore, four potential mark locations. Thus, the total number of marks needed to transmit a given number of message bits would be reduced by a factor of two. By the same token, the decoder would have to examine four mark locations at each node in the decoding tree but there would only be half as many nodes, meaning that the number of hash calls would be expected to remain about the same.

However, because each node requires the examination of more potential paths for further decoding, the opportunities to spawn hallucinations increases resulting in a corresponding decrease in the critical density. The number of hallucinations in steady state as a function of the number of message bits, B, corresponding to each call to the hash function is

$$H_{SS}(B) = \frac{2^B - 1\mu}{1 - 2^B\mu} \tag{7.16}$$

As before, the critical density corresponds to the denominator going singular, or

$$\mu_C(B) = \frac{1}{2^B} \tag{7.17}$$

Therefore, a quadrature system that encodes two bits in each mark would have a critical density of only 25%, which is below the stipulated density of the threat model. Of course, the threat model is admittedly arbitrary and not set in stone. Recall that limit of the present threat model was motivated by setting the number of working hallucinations to one per valid message and then asking if this was a reasonable threat model in practice. Using a similar approach here, the limit of the threat model would have to be reduced to about 14%, which is hard to argue is reasonable. But if, say, a threat model limit of 20% were considered reasonable, then the decoder would only have to be able to handle three hallucinations per valid message, which is almost certainly reasonable. None-the-less, M-ary BBC is probably not a viable candidate, at least by itself, for a jam-resistant waveform.

7.2.3 Multimark BBC

An alternative to encoding multiple message bits in each mark is to use multiple marks to encode each message bit. This is potentially useful for two purposes: First, if each mark is required to be present, it increases the codec's critical density. Second, it offers a way to tolerate mark errors by only requiring a minimum subset of the marks associated with each bit be present, though this tends to reduce the critical density. In effect, the set of marks constitute a *composite mark* that has a particular probability of being detected and can be considered equivalent to a single mark in a packet having that density, at which point all prior results concerning critical density and hallucination workload apply.

In the general case, each path is represented by Q-of-Y marks, meaning that the encoder generated Y marks of which Q must be detected. The likelihood that any given mark will be found is, as always, the packet mark density, μ. The probability that a hallucination will be spawned (or extended) along a particular non-message

Table 7.3: Multimark BBC parameters for up to three marks per bit (MLR for PLR = 1% for 1000-mark codewords).

Q	Y	$\mu_C(\%)$	MLR (ppm)
1	1	50%	10
1	2	29%	3170
2	2	71%	5
1	3	21%	21580
2	3	50%	1831
3	3	79%	3

path is simply the probability, μ_e that there will be Q or more marks out of Y, thus

$$\mu_e(Q,Y) = \sum_{i=Q}^{Y} \binom{Y}{i} \mu^i (1-\mu)^{(Y-i)} \tag{7.18}$$

This effective mark density, μ_e, can then be substituted into 7.15 for critical density or 7.10 for the hallucination workload (without interstitial checksum bits). Table 7.2.3 shows the critical density and also maximum mark loss rates (MLR) corresponding to a packet loss rate of 1% for 1000-mark codewords.

As an example, in 2-of-3 multimark, the hash function is used to determine three mark locations for each mark but the decoder only requires that two be found. For this case

$$\mu_e = \mu^2 (3 - 2\mu) \tag{7.19}$$

Somewhat surprisingly, for this configuration the increase in critical density due to using more marks per message bit exactly cancels the decrease in critical density due to allowing some marks to be missed and, hence, the critical density remains 50%, but the hallucination burden at the 33% threat model limit is reduced to to 0.54 hallucinations per valid message. If a hash function is used that produces all three mark locations in one call (which most hash functions would be capable of), the only significant processing cost lies in the memory accesses associated with checking

the mark locations. However, since there are only slightly over half the number of hallucinations per legitimate message and since the third mark location would seldom need to be checked, the actual increase in processing costs should be marginal.

The real potential gain offered by multimark BBC comes from the significant increase in the tolerance to mark errors. As an example, in Standard BBC (which is identical to 1-of-1 multimark), a mark error rate of less than 10^{-5} is needed to achieve a packet loss rate of less than 1% for thousand bit messages. However, using 2-of-3 multimark, the same packet loss rate can be achieved with a mark error rate of 0.18%.

The cost of using multimark BBC is that the energy budget per bit must be spread out over more marks. For n-of-3 multimark, this reduces the effective SNR-per-bit by nearly 5dB while simultaneously increasing the codeword density. Whether the improvement in the allowed MLR is able to overcome this cost must be calculated (the approach for which is developed in Chapters 9 & 10. It is likely that the outcome of this analysis will be different for different types of marks and for different detection methods.

A subtle benefit of multimark BBC is that, while we must assume that the attacker knows how many marks are transmitted per bit (since the receiver must also know this), the attacker need not have knowledge of how many marks the receiver will require in determining the presence of the composite mark (since the transmitter does not need this information). Thus, for example, the receiver could require only two of the marks be found and keep the threshold so as to produce a received packet density of 33%, as with standard BBC, or it could require that all three marks be present and set the threshold to achieve a received packet density of, say, 70%. In addition, it could require only a single mark and set the threshold to hold the received packet density to only 15%. In fact, BBC decoding lends itself to highly parallelized

Table 7.4: Block-Oriented BBC Codec Parameters.

L	=	message length (bits)
F	=	expansion factor
C	=	codeword length (bits)
K	=	number of terminal checksum bits
S	=	number of interstitial checksum bits per fragment
D	=	number of information bits per fragment
B	=	number of padded message bits per encoding block
Y	=	number of marks generated per encoding block
Q	=	number of marks that must be found to declare a block found

implementations and it is quite practical to configure the system to use all three thresholds and perform decoding under all three options simultaneously.

7.2.4 Block-Oriented BBC

Standard BBC plus all of the enhancements described previously can be combined into a single codec description, known as a block-oriented BBC codec. that encompasses each variant as a special case. The parameters are shown in Table 7.2.4. Note that, since $C = LF$, only two of these three quantities are actual parameters with the third being defined by the other two.

The codec works as follows:

1) The L-bit message is broken into a sequence of D-bit fragments.

2) Each D-bit fragment is prepended with S checksum bits.

3) K checksum bits are appended to the end.

All of the above, together and in order, constitute the padded message.

4) Encoding prefix of the padded message that is a multiple of B bits is hashed.

5) Each hash output contributes Y marks to the C-bit codeword.

For this to work most cleanly, L should be divisible by D and length of the padded message should be divisible by B. However, this is not absolutely required as long as unambiguous rules exist on how to cope with the exceptions.

7.3 Practical Considerations

7.3.1 Bookend Marks

Like any communication system, the receiver has to synchronize to the transmitted signal. This places a quiescent burden on the decoder because each symbol period requires the decoder to evaluate whether a packet exists starting with that symbol. For Standard BBC, this requires the decoder to begin the decoding process and continue it until the decoding tree dies. The expected number of calls to the hash function, assuming there is no valid packet at that location, is given by

$$N_Q = \frac{2}{1-2\mu} \tag{7.20}$$

At the edge of the threat model, namely packet densities of $\mu = 33\%$, this would result in an average of six calls to the hash function per symbol period, independent of the mark density in the bitstream being processed. While this doesn't sound excessive at first glance, comparison to the active burden (the number of calls involved to decode packets containing valid messages) reveals its true nature. For this, we assume a continuous, back-to-back string of packets containing one message each. For C-bit codewords containing an L-bit message with a K-bit checksum, the number of calls to the hash function at this same mark density would be $2(2L+K)$ every K symbols. Ignoring the minor impact of the checksum bits and using thousand bit messages and million bit codewords, this results in an average active burden of only four hash calls per thousand symbols, making the quiescent burden 1,500 times the active burden. Even if the mark density was the result of each packet containing approximately 400 messages, the quiescent burden would still be nearly four times the active burden.

A simply way to significantly reduce the quiescent burden is to place a mark in the leading position of each codeword. Thus, the codec would only initiate the decoding

process when it actually sees a mark. At the edge of the threat model, this would be 33% of the time, thus the quiescent workload at this mark density would be reduced to just two hash calls per symbol. A comparable additional improvement can be gained by placing a mark in the trailing location of the codeword as well. Now the decoder would require the presence of both of these "bookend" marks before initiating a decode, reducing the quiescent burden to only 0.67 hash calls per symbol. While this would still put the quiescent-to-active burden at 167, that represents a significant improvement.

Of course, if there were a need to reduce the quiescent burden further, the number of bookend marks could be increased (and there is no requirement that they be placed at any particular location within the codeword, as long as the protocol specified the locations. In order to reduce the quiescent burden to below the active burden (when operating at the threat model limits) there would only need to be seven bookend marks.

Bookend marks are not completely cost-free, however. While no call to the hash function must be made, the packet must still be examined in each of the locations. However, since this search is short-circuiting, meaning that the first bookend mark not found terminates the search, the burden does not grow excessively as additional bookend marks are added.

7.3.2 Jitter and Symbol Phase Compensation

Up to this point, it has been assumed that the oscillators in the transmitter and receiver are ideal and perfectly matched; furthermore, it is assumed that the receiver is perfectly aligned with the received symbols. Clearly these are both unreasonable assumptions. This is particularly the case here since this is a completely asynchronous protocol. In order to avoid framing errors with such a protocol, which occur when the last bit in a packet is too far out of alignment with where the receiver expects

it to be, a common rule of thumb is that the sending and receiving oscillators must match within about one-tenth of a part per the length of the packet. Beyond this, it becomes highly likely that intersymbol interference will become intolerable. Thus, for a thousand bit (and, hence, thousand symbol) packet, it would be expected that oscillators rated for no more than about 500ppm would be necessary at both ends just to avoid framing errors. Details of the modulation scheme would likely tighten this requirement even further. Since BBC uses packets that are typically on the order of one million bits, conventional wisdom would dictate oscillators that are better than 0.5ppm, which exist but are not inexpensive. Even if the oscillators are well matched, on average, many noise sources in the transmitter (and possibly in the channel and receiver, as well) would contribute to uncertainty, called jitter, in the actual location of any given symbol boundary, which would also result in intersymbol interference.

Fortunately, the high asymmetry associated with a concurrent channel offers a simple way to sacrifice only a modest amount of performance to overcome significant amounts of timing mismatch. First, let's consider symbol synchronization; even when the oscillators match perfectly, there is a random phase between the symbol boundaries that must be accommodated. Any mismatch in symbol alignment will result in intersymbol interference. However, in a highly asymmetric channel that is concurrently coded, significant intersymbol interference is inherently tolerated. For instance, consider a simple radiometric detector. Normally, such a detector would look at the total energy between time nt and $nt + T_S$ (where T_S is the symbol duration) to determine whether a mark exists at time slot n. If the received mark straddles the symbol boundaries such that any appreciable portion of it were in an adjacent time slot, then a traditional receiver would be highly likely to make a bit error in one or even both of the time slots. However, in a concurrent receiver, the detection threshold will normally be low enough that it will still have a very high likelihood of declaring a mark in whichever time slot has more than half of the energy while

the other slot will likely have a significant chance of making a space error and falsely declaring a mark. Thus, while an error is still likely to occur, it is highly probable that it will be one that the decoding algorithms are specifically designed to deal with.

Although the basic behavior of a concurrent channel offers considerable protection against symbol misalignment, deliberate steps can be taken to improve it. One way would be for the transmitter to simply increase the width of the marks that it transmits, though just how simple this is in practice depends on the exact mark waveform that is used; in a simple pulse-based system using gated noise or an unmodulated carrier, it would be trivial. Consider a transmitter that transmitted pulses that were $2T_S$ long for each mark. If the receiver is using a radiometric detector that is integrating over T_S, then no matter how misaligned the receiver's symbol clock is to the received signal (assuming no oscillator mismatch, which can be compensated for as well), the full energy of a mark (i.e., the energy expended over T_S by the transmitter) is present during at least one of the packet frames that will be examined. In all likelihood, the packet will be received two, or even three, times in successive frames, but multiple reception of packets is something with which all transport protocols operating over unreliable channels must contend [77]. Their are three costs for using this approach: first, the transmitted packets have a higher mark density meaning that that, in principle, the jammer has to place marks in fewer locations in order to reach their target packet density. However, the transmitted packet is going from something on the order of 0.1% density to 0.2% while the jammer now needs to fill 33.1% of the packet with marks instead of 33.2%, so this, by itself, has negligible impact. Second, the transmitter must now spread the energy budget for a mark across multiple symbol periods which, while keeping the total energy expenditure the same, reduces the effective energy per mark and, correspondingly, the jammer's required expenditure. Alternatively, the transmitter can increase their energy budget to keep the effective mark energy the same, but the jammer can then keep their energy expenditure the

same. In either case, the performance curve shifts in favor of the attacker. Finally, the decoder will almost certainly have to at least partially decode additional messages and will likely have to authenticate some duplicate messages; since any time spent doing that becomes unavailable for processing attack packets, the performance curve is again shifted toward the attacker. On the other hand, the likelihood of a packet being lost due to symbol misalignment is drastically reduced and, in addition, each packet is now effectively being transmitted multiple times and all copies must be missed for the packet to be lost.

Another corresponding method would be for the receiver to widen the window over which it looks for marks. For instance, when looking for a mark at location p, it could simply examine both p and $p+1$ and consider the mark found if there is a mark in either location. If the packet data available to the decoder has been hard-decided, meaning that it has already been declared a mark or a space in each packet element, then something like this is the only option available to the receiver. However, if the signal amplitude at the discriminator input have been stored, then the receiver can use much more creative decision algorithms. For instance, a mark could be declared if the sum of the signals for p and $p+1$ is above the threshold. This method has the particular advantage that it requires nothing different on the part of the transmitter to implement. As a result, unless the decision is a fixed part of the protocol, the jammer has no way of knowing whether this technique is being used or not, which complicates their jamming strategy. On the other hand, when used, the same processing steps are applied to both the legitimate marks and the jammer's marks, meaning that the effective packet density has increased and, correspondingly, the effective critical packet density has decreased.

Not surprisingly, all of these reduce the effective degree of jam resistance; however, that performance loss also brings with it gains, some of which are hard to quantify. First off, the lower packet loss rate will reduce the benefit surrendered to the attacker

and may, by itself, tip the trade back in favor of the legitimate parties. Beyond that, in traditional systems, a jammer has many different ways to attack a signal other than just increasing the noise floor to cause bit errors. They can attack the receiver's synchronization subsystem, which provides the ability to servo its oscillator to match the incoming signal, to align to its symbol boundaries, and to detect packet boundaries. In a concurrent channel, things can be kept so simple that they are, in essence, too stupid to be fooled. Thus, as with traditional spread spectrum, whether exploiting mark expansion is worthwhile depends on the cost-benefit tradeoffs involved.

7.3.3 Oscillator Mismatch Compensation

It's one thing to be largely immune to jitter and symbol phase errors, but it is quite another to deal with framing errors resulting from mismatch between the sender's and receiver's underlying oscillators. For instance, if the two oscillators are mismatched by just 10ppm, then the final symbol of a one million bit codeword would be received ten symbol periods early (or late) as seen by the receiver. In most traditional systems, this would be dealt with by servoing the receiver's oscillator to the received symbol using some artifact embedded in the symbol stream. This might be an embedded clock or it could be a message preamble. Unfortunately, this is not a viable option for a concurrent channel for several reasons. First, the whole point is to be able to accommodate multiple signals arriving concurrently and each signal is going to have its own oscillator mismatch. Second, concurrent channels lend themselves to sparse, unipodal signalling schemes which do not lend themselves to embedded clocks or preambles. Finally, any mechanism used to tune the receiver's oscillator based on the detected signal invites an attacker to devise a signal that leads the receiver astray. None-the-less, the issue cannot simply be ignored.

In the case BBC-based systems, packets must be entirely buffered in memory before they can be decoded. This permits a way of tolerating effectively arbitrary

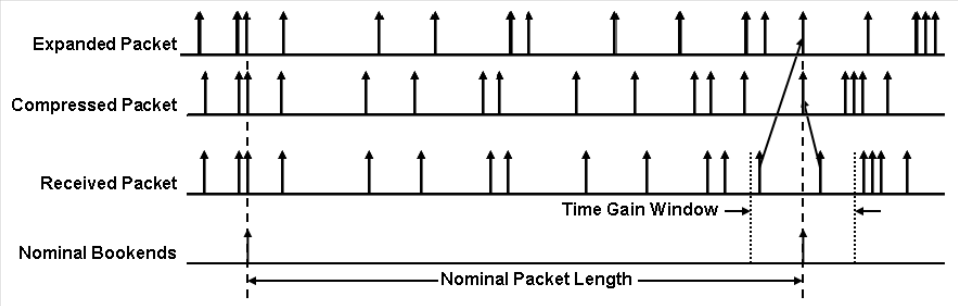

Figure 7.7: Compensating for oscillator mismatch.

levels of oscillator mismatch using a very simple artifact. The basic concept is that each time a mark is detected, it is assumed to possibly be the leading bookend mark of a packet. Normally, the decoder would then look for the trailing bookend mark and only attempt to decode the packet if it is found at the expected location. To compensate for oscillator mismatch, the decoder instead examines a window centered at the expected location, in time, of the trailing bookend mark, as shown in Figure 7.3.3.

The size of the window is determined by the maximum amount of mismatch that is to be tolerated. Any marks found within that window are assumed to potentially be the trailing bookend mark from a slightly mismatched transmitter. For each such mark, the packet is decoded by calculating an adjusted mark location based on the amount of mismatch indicated. As an example, consider a codec that is working with million bit codewords and must tolerate up to 100ppm of oscillator mismatch. The decoder would need to examine a window that is basically 200 bits wide and, at the limit of the threat model, would find approximately 67 marks. Decoding all of these potential packets would increase the quiescent burden on the decoder by a factor of 67 which, while significant, is not by any means prohibitive. Most of this burden can be eliminated with an additional three or four bookend marks.

7.3.4 Concurrent Hash Functions

Just as not all hash functions are suitable for cryptographic applications, not all hash functions are suitable for use in a concurrent code. In general, the properties of a "good" hash function cannot be divorced from the application that it is intended to be good at. In the case of a *concurrent hash function* (namely, a hash function used in a particular concurrent codec), the goal is for it to be computationally infeasible of designing effective attack packets where an effective attack packet is one that places a disproportionately large burden on the decoder. The properties required and/or desirable for a concurrent hash function for a BBC-based concurrent codec have been laid out in the paper that introduced the Inchworm concurrent hash function [79].

One of the characteristics that a concurrent hash function must possess in order to avoid a known theoretical attack is an internal state that is large relative to the number of invocations required to encode a message, with a factor of two being considered adequate. Thus, for a one thousand bit message encoded with Standard BBC, the internal state should be at least two thousand bits. Since all mainstream cryptographic hash functions have internal states considerably smaller than this, they all possess this theoretical (though probably not practical) weakness.

Because a concurrent hash function is going to be called on the order of once per symbol in the data stream, it must also be computationally efficient. In practice, this means it should be both incremental and reversible. Incremental means that, given a hash function that has just hashed a particular message prefix, it should require constant effort to hash a new prefix consisting of the prior prefix plus an additional bit. Conversely, reversible means that it should require constant effort to return the hash function to the state corresponding to the prefix that is one bit shorter than the one most recently hashed. Obviously, the amount of that constant effort must be minimal. Currently, the recommended concurrent hash is Glowworm, which should be published in the near future. In the meantime, the Inchworm hash

is a good alternative, since the weakness found in it (and described in the upcoming Glowworm paper) is theoretical and not practical. Inchworm and Glowworm are both speed and space efficient in both software and hardware implementations.

7.3.5 Running Statistic Threshold

If a concurrent channel is unipodal, there is no natural level at which to set the detector's threshold. While manually setting the threshold is unlikely to prove satisfactory in most applications, using an automatic method risks creating vulnerabilities that an attacker can exploit. For instance, if the average packet density is tracked using a low pass filter and the threshold adjusted to maintain the packet density near 33%, on average, then an attacker could flood the spectrum with energy for a short period of time to force the receiver to raise its threshold knowing that, even once they stopped transmitting, the threshold would require some amount of time to recover and, in the meantime, would still be high enough to raise the mark error rate unacceptably. To minimize this, the response time of the threshold adjustment circuit would need to be very short, but this invites the attacker to try to cause the receiver to produce an erratically changing threshold with the goal of producing mark errors over short time spans that are repeated at roughly the period of a packet.

The underlying problem with using an average density to set the threshold is that the threshold for the packet currently being decoded is, in part, dependent on signals received that are not part of the packet. Thus, an attacker's influence extends beyond the time span during which they are actively emitting. The obvious solution, then, is to have the receiver set the threshold at the value that is appropriate for the packet presently being decoded based only on the signals that are a part of that packet. For instance, if each packet were decoded using a threshold that resulted in that packet's mark density being at or near the threat model limit, then the attacker can only influence a packet with signals that directly interfere with that packet. However,

implementing such a scheme presents two significant challenges. First, what this entails is determining a threshold that corresponds to a certain percentile, which involves ordering the data. This ordering must then be updated at each time step (i.e., for each received symbol). Second, the symbol received at a particular time is not just a part of a single packet; it belongs to as many potential packets as there are bits in a codeword. Thus the value of the symbol must be determined according to that many different thresholds applied to the underlying signal. As a consequence, it is not sufficient for the discriminator to produce a simple two-level output. Instead, a multi-leveled value must be stored in the buffer and then compared to each packet's threshold in turn.

Accomplishing the latter is a straightforward matter of storing the value that will be presented to the discriminator instead of the value at its output. The former challenge, however, requires a much more elegant solution in order meet the limited computational resources that will most likely be available. What is needed is a processing block that can accept the value of the next symbol's amplitude and then produce the smallest threshold for which no more than 33% (or whatever statistic is used) of the last C symbols received (where C is the number of symbols in a packet) have amplitudes greater than that threshold. This is referred to as a "running statistic" module (if the statistic sought is the 50%-tile, then it would most likely be known as a "running median" module. At first glance, this would appear to require ordering all of the symbol amplitudes in a packet. At this point, determining the appropriate threshold is trivial. However, when the next symbol arrives, the amplitude of the oldest symbol must be removed, the amplitude of the new arrival added, and the ordering of the list updated. With million-bit packets, performing these tasks at the symbol arrival rate is anything but trivial.

The key to an efficient implementation is to recognize that the list does not have to be completely ordered. All that is required is to partition the set of values into the

largest 33% and the smallest 67%, to always be able to quickly identify the smallest and largest values in each partition, respectively, and to be able to replace the oldest item with the newest item and heal the relationships efficiently. This can be done using a data structure known as a heap or, alternative, a priority queue. A maximal heap is a tree structure in which each parent is at least as large as any of its children; by contrast, in a minimal heap each parent is at least as small as any child. Thus, the root node of a maximal heap is guaranteed to be the largest value in the heap while the root node of a minimal heap is guaranteed to be the smallest.

Consider a maximal heap that is validly organized and assume that a randomly chosen node is updated with a new value. In all likelihood, the heap is no longer validly organized; however, it can be "healed" using only vertical swaps with parents or children and, more importantly, those swaps will always be in the same direction. For instance, assume that the new value is greater than the value of its parent. Then it is known that the new value must also be at least as large as either of its children, so they do not even need to be checked. Instead, the new value is swapped with its parent. There is no need to check if the old parent that has moved downward is at least as large as both of its children since it is known that it was at least as large as the original parent of both. This upward migration continues until either a point is reached in which the new node is no longer larger than its current parent or until the new node reaches the top of the heap. Conversely, assume that the new node is smaller than the largest of its two children. In this case the new node must be swapped with the largest child. Using the largest child guarantees that the new parent has the proper relationship to its former sibling. Similar to before, there is no need to check if the new parent is larger than its new parent since this is guaranteed by the original relationships. Again, this downward migration continues until a node is reached in which the new values is as least as large as the largest of its children or it has reached the deepest level of the heap. In the case of a binary heap, there are

$1 + \lfloor \log_2(N) \rfloor$) levels where N is the number of nodes in the heap and $\lfloor x \rfloor$ returns the largest integer not exceeding x. The maximum number of swaps needed to heal a heap is therefore one less than this, or simply $\lfloor \log_2(N) \rfloor$).

Now consider a combined structure in which a maximal "top" heap containing $C/3$ nodes shares a root node with a minimal "bottom" heap containing $2C/3$ nodes where, again, C is the number of bits in a packet. Requiring symbol amplitudes to exceed the value stored in the root node will result in packet densities as close to 1/3 is possible without exceeding it. For the case of a one-million bit packet, this would put 333,334 nodes in the top heap and 666,666 nodes in the bottom heap. The top heap would have 19 levels while the bottom heap would have 20. Because of the shared root node, the maximum number of swaps required to heal the combined structure would be 37. However, most updates will involve replacing a value that is fairly deep in one heap with a new value that will need to end up fairly deep in that same heap, thus the actual umber of swaps per healing should average around half of the maximum, or about 18 in this case.

There are several bookkeeping issues that must be dealt with efficiently in order to maximize the throughput, but careful construction using mirrored arrays and indirect referencing can result in a very lean process [80].

Chapter 8

Optimal Jamming Signal for CCSS

In analyzing the jam resistance of traditional spread spectrum systems, it is commonly assumed that Kerckhoff's principle (a.k.a., Shannon's maxim) applies and that the adversary has complete knowledge of the system except for the keys. In particular, it is assumed they have full knowledge of the waveform in general including things such as frequency pallet, modulation scheme, and symbol duration. In addition, they know how the signal is constructed and how it is processed. Finally, it is even assumed that they know the signal-to-noise ratio (SNR) of the legitimate signal at the receiver and that they have the ability to control their own signal's SNR at the same receiver. However, without knowledge of the keys (the spreading code, in the case of a spread spectrum system), they are unable to inject coherent content into a signal; rather, they attack the waveform in order to inject effectively random information errors. Since this is generally the only kind of jamming that is considered, it is not given a particular name. However, in concurrent code spread spectrum, the adversary does know the spreading code and therefore can inject coherent content into the received waveform. This gives them additional jamming strategies to exploit and therefore it is useful to categorize them; therefore, traditional jamming will be referred to as "waveform jamming" while jamming aimed at exploiting knowledge of the spreading

code will be termed "protocol jamming." Because a jammer can use either approach separately or a combination of the two, thus all three possibilities must be considered and, where possible, mitigated and/or analyzed.

8.1 Threat Model

If practical, a hostile jammer facing a CCSS system would transmit a waveform that only resulted in the receiver committing mark errors; erasing only a few marks would constitute a devastatingly effective attack. However, as noted before, accomplishing this by attempting to cancel out the legitimate signal, at the receiver, is not a trivial undertaking. The more likely outcome of such an attempt is that the attacker's signal would actually make it more likely that the presence of a signal would be sensed and a mark detected.

None-the-less, the jammer has working in their favor the fact that it is quite easy for them to add marks to the received signal stream, forcing the decoder to perform additional work. At some point, if they are able to inject enough marks to exceed the decoder's critical mark density, then the workload will explode exponentially and the channel will be jammed. However, this is fundamentally no different than any other form of spread spectrum – if the jammer is capable of and willing to commit sufficient energy to the attack, they will win. From a practical standpoint, other considerations must impose a limit on the amount of energy the attacker is willing to commit; for that, we must consider the adversary's risk assessment. For example, if the anti-jamming is intended to protect domestic communications for first responders, then the goal is to force the attacker to use enough energy so that law enforcement can detect and localize the adversary, who would then face potential fines and imprisonment. In a tactical military setting, the goal is much the same, though the response tends to be somewhat more direct and permanent. With that in mind, a more complete goal

statement for an anti-jamming technology would be to provide enough resistance to hostile jamming so as to keep the communications channel open in the presence of the strongest attack the adversary is likely to be capable of and/or willing to execute.

For CCSS, a somewhat arbitrary limit to the threat model of 33% mark density has been established [24]. With a packet mark density of 33%, the decoder is expected to have to perform about twice the number of computations that it would in a noise-free environment (provided the extra marks are randomly placed). The argument that this is a reasonable limit goes as follows: Given a detection threshold suitable for detecting legitimate marks, there is some minimum amount of energy that, if committed to a jamming attack, would flood the spectrum with sufficient energy to turn every space into a mark. The network defenders, whether they use warrants or missiles, are either capable of localizing and neutralizing an attack of that level or they are not. If they are not, then the attacker is going to win. But if they are, then they are likely still capable of doing so against an attack level at one-third that energy or it should be possible to raise the power of legitimate signal, by at most a factor of three, to make it so. Given this, the channel will remain open at that level of attack and the attacker loses. With this in mind, our analyses of protocol attacks will assume that an attacker is only willing to accept the risk associated with turning one-third of the spaces into marks.

8.2 Protocol Jamming

In analyzing the impact of protocol jamming, we will assume that the attacker can inject into a packet specific marks at locations of their choosing and can bring the total packet density to the extreme edge of the threat model, which is generally taken to be 33% (and which we will assume from this point on, even though it can always be set to a different level). Furthermore, we assume that the attacker can accomplish

this without any risk of retaliation. These assumptions are quite reasonable if the receiver is using a running statistic threshold because it is assumed that the threshold is chosen so as to maintain the packet density at 33% even when no jamming signal is present. Thus, the attacker must only broadcast marks that are marginally above the noise floor to ensure that they detected and we are granting them the ability to do this without risk. On the other hand, they cannot inject any more marks because the running statistic threshold will reject all but the 33% of the marks with the highest energy. Thus, to have any effect beyond injecting enough chosen marks to bring the packet density to 33%, the attacker would have to increase their energy output enough so as to place at least some of the legitimate marks below the top third. At this point, they are conducting a waveform attack (in additional to a protocol attack) and the waveform aspect of it is treated separately.

Because the receiver has the ability to force the packet density to a value comfortably below the critical density, an attacker is unable to present the codec with an exponentially growing decoding tree. However, they can still strive to use their mark budget so as to create the most computationally expensive tree possible. There are two targets of such an attack; the first is to produce a tree with the maximum number of nodes while the second is to produce a packet that the greatest number of final messages that must be authenticated. Which of these is the more effective of the two depends both on the relative success of the packet at achieving each result and the relative ability of the receiver to cope with each. Each will be considered separately.

Research on crafting optimal attack packets is ongoing and and outside the scope of this dissertation. While it therefore cannot be guaranteed that a hypereffective attack packet that forces an exponentially large decoder workload cannot be formed, a security proof exists [81] showing that, for a random hash function selected from among all possible hash functions, that the probability of such a hypereffective attack

packet existing is vanishingly small and, at the risk of extreme understatement, the probability of finding it is substantially less. However, it is known that hash functions can be constructed for which such hypereffective attack packets are trivial to form. Fortunately, the adversary has no say in the choice of the hash function, they merely know what choice was made. Thus they must devise a means of searching for effective attack packets against the specific hash function in use. Given the search space involved, a brute force search to find the global maximum is almost literally unimaginable. To date, the most effective approach has been to build packets mark-by-mark using a gradient descent approach to find packets that, locally, maximize the decoder workload. Such packets are generally a few times more effective than simple multi-message attack packets and, therefore, easily countered with only a modest increase in decoder cost and complexity.

8.2.1 Random Mark Attack

The simplest protocol attack is simply to inject random marks into the received packet. For a BBC codec configured to work with L-bit messages and an expansion factor of F, the number of attack marks, N, that the attacker should be able to add to a packet to achieve a desired density can be obtained by inverting (7.6).

$$N = -FL\ln(1-\mu) \tag{8.1}$$

For our standing example of a Standard BBC codec, namely 1000-bit messages, 30-bit checksums, and an expansion factor of 1000, and with an attack limit of 33%, this gives the attacker 405,465 marks to work with; they may have more or less, in practice, since this is the expected number of independent marks they could add to million-bit attack packet before reaching the 33% limit.

However, if a running statistic threshold is in use, then this is a singularly ineffective attack since random noise is already accomplishing the same thing. Furthermore, it has already been shown that a random mark density of 33% only forces the decoder to perform twice the computation that it would have performed in the absence of any noise at all.

8.2.2 Random Message Attack

The next simplest attack is to construct an attack packet consisting of randomly chosen messages. Each such message not only serves as a source for the generation of hallucinations, but also as a final message that must be discriminated against via authentication.

The nominal number of messages that the attacker could add would be the packet mark density times the expansion factor, or μF. The presence of the checksum bits drives this down while the mark collisions tend to drive it up. The number of independent marks that the attacker has to work with is still that given by (8.1). Combined with Eqn. 7.8 and taking into account checksum bits, we have the number of attack messages, M, as

$$M = -\frac{F\ln(1-\mu)}{1+\frac{K-\log_2(M)}{L}} \tag{8.2}$$

While this can't be solved in closed form, the base-2 log of the number of messages is generally small enough relative to the message length that it can be ignored (this is certainly true in any practical codec).

Again turning to our standing example of a Standard BBC codec with 1000-bit messages, 30-bit checksums, an expansion factor of 1000, and an attack limit of 33%, this yields an expected load of 393 messages. Adding in the impact of the

hallucinations at this density, and the decoder would be expected to perform about 800 times the computational work compared to the zero noise case.

In practice, the attacker needs to use a few less than this because they do not want their marks competing with the legitimate marks for inclusion. The reason is that a legitimate message would exclude about a thousand of the attack marks (remember, the energy in the attack marks is assumed to be just above the noise floor) which would likely destroy most, if not all, of the attack messages at some point during the decoding process. But this could be easily addressed by using slightly higher energy for most of the attack messages and slightly less for the last few, which would be sacrificed in favor of the legitimate marks.

8.2.3 Hallucination Attack

If the goal of the attacker is to maximize the codec's hallucinogenic burden, then one step in that direction is to use a "bushy-headed" attack which generates as many message lines in the codec as possible as early as possible. It also makes no attempt to support these messages once the terminal checksum bits are reached. However, this results in only a marginal increase in the number of messages compared to the complete random message attack, especially if the random message attack simply stops encoding the random messages upon reaching the terminal checksum bits.

This is one place where interstitial checksum bits can play a useful role. In essence, each such bit forces the attacker to insert an attack mark that can produce no hallucinations while, at the same time, reducing the hallucinations that are already present.

The approach described above is simple to implement, but not necessarily anywhere near optimal. Although highly unlikely, it is always possible that there exists sets of marks that constitute unusually effective attack packets. Assuming the hash function chosen is sound, it should not be possible for the attacker to analytically construct such packets and it is clearly impossible to even begin to explore the space

of all possible attack packets in search of the globally most effective one. But there are numerous ways to find the most effective attack packet in a small neighborhood of the packet space. One possibility would be to use a gradient descent (or, perhaps more correctly, a gradient ascent) approach. For instance, a "seed packet" could be constructed with one or more messages (or message fragments) and then the size of the decode tree determined by actually decoding the packet. Then a new mark could be placed at each empty space of the packet, in turn, and the new decoding effort determined for each location. The mark would then be frozen at whichever location produced the greatest increase. This process could then be repeated until the desired attack packet density was achieved. There are almost unimaginable variations on this theme, such as periodically removing the least useful marks. Simulations to date, on packets of up to a few thousand bits, indicate that packets several times more effective than random messages can be constructed, but no highly effective packets have been discovered.

8.2.4 Authentication Attack

If the attacker wishes to flood the receiver with the maximum number of valid messages that must be individually authenticated, then they must intentionally inject all of them since they cannot count on any hallucinations surviving all of the terminal checksum bits. The notion of a bushy-tailed packet was discussed in Section 7.1.4. This is the type of packet that will produce the greatest expected number of final messages at a given mark density.

Once again, the attacker has the mark budget given by (7.6) with which to construct their attack. For a Standard BBC codec configured to work with L-bit messages and K-bit checksums, they must allocate their marks three ways to produce M final messages: They must use $L - \log_2(M)$ marks to propagate a message from the start to the point at which the bushy tail begins. They must use $\approxeq 2(M-1)$ marks to

create the tail, and they must use KM marks to sustain them through the terminal checksum decoding. Thus, the number of final messages that can be forced is

$$\begin{aligned} M &= -\frac{-L(F\ln(1-\mu)+1)+\log_2(M)+2}{K+2} \\ &\approxeq -\frac{C\ln(1-\mu)}{K+2} \end{aligned} \tag{8.3}$$

Again turning to our standing example of a Standard BBC codec with 1000-bit messages, 30-bit checksums, an expansion factor of 1000, and an attack limit of 33%, this yields an expected load of 12,640 messages. Using the simpler approximation, a value of 12,671 is obtained, showing that it is likely to be a sufficient approximation for most purposes.

This is a potentially serious attack that could increase communication latency to an unacceptable level as the higher levels in the stack become overburdened performing message authentication. One possible approach is to leverage the fact that such an attack packet is unlikely to align with a legitimate packet and thus setting a threshold on the maximum number of messages that any given packet can produce before being abandoned is unlikely, by itself, to significantly increase the rate of lost legitimate messages. However, it must be kept in mind that imposing such a limit means that the attacker can compose multiple overlaid attack packets each producing the allowed limit. For instance, using the prior example, if the decoder were set to abandon packets once they reach 128 messages, then the attacker would only need to use 4191 marks to create their attack packet. Since this is only slightly over 1% of their mark budget, theY could repeat the packet (or other, similar packets, at a rate such that 97 attack messages began, on average, during the length of a packet. While this increases the likelihood that a legitimate message will be masked by an attack packet by a factor of nearly one hundred, the chances of such a collision are still about 10,000:1. It also reduces the total number of false messages they can force

the user to authenticate to about 9,700, which, while not decisive, is none-the-less nearly a 25% reduction.

Placing such an arbitrary limit on the decoder can be removed by prepending each message with a cryptographic hash of the message. Since such a hash would be required as part of signing the message anyway, the computational cost on the sender's part is negligible. Using a 128-bit hash would increase the bandwidth requirement and also the codeword density by ten to fifteen percent in our typical example, all else being equal.

A potentially more efficient alternative to mitigate the attack is to salt each message with a random prefix and then use some means of throttling the decoder output by limiting the number of messages that can share a prefix. Taken to the extreme of accepting the first message produced by any given prefix, the attacker must now support each of their attack messages throughout the entire decode process which limits them to about 400 such messages in our example (which they can put in one packet or spread out over multiple partially overlaid messages). If a sixteen bit salt is used, then even if all of the attack messages are in a single packet and even if they succeed in aligning this packet with a legitimate packet, their is less than a 1% chance of colliding with the legitimate message's random prefix. A much more sophisticated approach is described in [82] in which both the decoder and the authenticator can gracefully terminate decoding of a packet prefix on an as-needed basis.

Even if a bushy-tailed attack is mitigated completely, the attacker can always force the receiver to authenticate full-length messages which, in our example case, is about four hundred messages for each legitimate message. This underscores the need for the adopted message protocol to incorporate an efficient means of validating and authenticating messages. It also reveals one disadvantage of using large expansion factors since the number of complete valid messages that the attacker can force per legitimate message is proportional to this factor. Of course, this merely means

that this issue becomes one of many interrelated constraints that the designers of a practical system must weigh.

8.3 Waveform Jamming

In analyzing the impact of waveform jamming, we concern ourselves only with the ability of the jammer to increase the level of random mark errors made by the receiver. The impact of space errors is analyzed as a form of protocol attack. In this research, the only waveform attack that was analyzed in detail was barrage jamming, the results of which are compared to similar analyses by Ziemer and Peterson [83] for traditional spread spectrum forms. As with other forms of spread spectrum, barrage jamming is very simple to implement, but it lacks effectiveness compared to the energy committed. Therefore, it is not expected that this analysis will provide a true picture of the performance of CCSS in a jamming environment, but rather a basis for a preliminary comparison of it performance against traditional spread spectrum systems confronted with similar barrage jamming attacks.

Like other forms of spread spectrum, it is expected that a jammer can improve their effectiveness by crafting jamming waveforms based on the specifics of the type of system they are attacking. One obvious improvement over barrage jamming is pulse jamming using a duty cycle equal to the receiver's packet density target. The idea is to force mark errors by simply presenting the receiver with enough false marks that are stronger than the weakest legitimate mark. If the receiver is going to set it's detection threshold such that 33% of the received bits are declared to be marks, then an attacker can broadcast noise with a 33% duty cycle with a peak power level comparable to that of the legitimate pulses. This alone should provide a 5dB increase in effectiveness over straight barrage jamming.

8.4 Composite Jamming

Assuming that an attacker is able to craft a signal that has a heightened probability of causing a mark error at a specific location, then an obvious question would be which marks in a packet should be the target of such an attack. Such an attack is a combination of a waveform and a protocol attack. The ability to craft such a signal is problematic and the chosen mark waveform and detection method will, of course, be designed to minimize the likelihood of success. But assuming that such a signal is discovered, then clearly the most valuable mark to go after would be the leading bookend mark since erasing this means that the packet probably will not even be detected, let along decoded. However, it must be understood that the ability to target any particular mark is extremely difficult since detecting the mere presence of a packet requires most of the packet to be received before it can even be detected. There is, after all, nothing to distinguish a bookend mark from other marks beside its relationship to the set of marks that follow it. Thus an attacker can attempt to predict when a message will be broadcast or detect a packet in transit early enough to broadcast an attack involving the remaining portion of the message. Predicting when a message will start is easily thwarted simply by having the sender dither their message start times within a suitably-sized window. Detecting a packet in time to overlay an effective attack on it is not trivial. Other than the leading bookend mark, there are no "start of frame" signals to detect. However, the locations of the marks associated with the leading bits in the message will belong to one member of a relatively small set. About half of those marks will appear in the latter half of the packet, and thus are of little utility in identifying a packet in time to overlay an attack signal. However, it must be assumed that the attacker will be able to prepare comparison masks of all of the leading prefixes so as to detect a packet in transit early on. Under these circumstances, an attacker can simply key their transmitter any time a packet is detected and as long as they can overlay the last third of the packet with

high-power noise, they will probably cause the packet to be lost. However, such a brute-force approach has a simple counter - the sender merely transmits a low level of random decoy packets at all times. If the attacker must go after each decoy in order to be sure they engage the legitimate messages, then their ability to detect packets early is of little benefit unless they can craft a low-power attack that causes mark errors. If they can, then the only marks that make sense to target are the trailing bookend marks. In particular, going after any particular message bit, including the checksum bits, past the first several is infeasible because determining the location to target requires the message thus far to be decoded which, in turn, is very likely to require marks that haven't been received yet. Going after the trailing bookend mark not only provides a static target, but also a target that gives the attacker the greatest time window with which to detect the packet and launch their attack. One method of mitigating the attack is to incorporate additional bookend marks throughout the packet (in which case, admittedly, the name "bookend" becomes a misnomer) and requiring only some minimum number of them to be detected.

Chapter 9

RF Implementations

At this point, readers should be reasonably comfortable that, given a suitable OR channel for transmission, a concurrent codec is capable of recovering large numbers of superimposed messages at a reasonable, though significant, processing burden. However, experience has shown that most people, at this point, remain very skeptical regarding the ability to fashion a suitable OR channel in the first place. The first part of this chapter therefore aims to address that skepticism by qualitatively establishing that such channels are reasonably easy to create and considering a number of possible options for doing so.

Having established the the necessary OR channel is conceptually feasible, we will then establish the groundwork for comparing CCSS to traditional forms by developing the Receiver Operating Characteristic (ROC) for a particular implementation, namely a pulsed-BBC CCSS system using a matched-filter detector. The ROC will then serve as the fundamental basis for the effective bit error rate (BER) curves developed in the next chapter. In order to work with the ROC curve, we will need to understand how, in a binary asymmetric channel, we should choose the optimal threshold setting and how it determines where on the ROC curve the receiver is operating. These issues will also be addressed in this chapter.

As already noted, the strategy employed here to obtain jam resistance is fundamentally tied to the error probabilities of the physical communications channel used. The goal is to achieve a "multiple access OR" channel[1] wherein the received bit stream is simply the bitwise-OR of all transmitted bit streams plus channel noise. In the ideal case the mark error probability (the probability of receiving a space when a mark was transmitted) will be zero, meaning that all transmitted marks will be received unaltered. However, in practice, a non-zero mark error probability must be tolerated.

To date, OR-channels have generally been considered only in terms of data storage structures in which codewords are combined via a bitwise-OR. These are the classic superimposed codes (by all their various names, such as Bloom filters) discussed previously. However, little attention has been given to physical channels that exhibit this behavior. This is not to say that such channels do not exist, but merely that this aspect of their behavior has apparently seldom before merited close examination. In the wired world one popular communications protocol used in many micro-controller based systems is I^2C (Inter-Integrated Circuit) whereby players are only allowed to assert marks; spaces only appear when no player is asserting a mark. This is generally implemented by using components having an open collector/drain output in a "wired-OR" arrangement. However, there is no effort to exploit this mark/space asymmetry to achieve jam resistance - in fact the protocol, like the majority of protocols, assumes that all players will play nice and obey the rules. In I^2C, the purpose of using a wired-OR connection is to attain collision detection and bus arbitration and the rules state that any player attempting to transmit a space but who senses a mark instead must assume that someone else is transmitting and immediately relinquish the bus. Such a protocol is, of course, trivially jammed by a malicious player with very little effort.

[1]This term appears to be the most widely used ([84] for instance), but others such as *multiaccess OR* [72], or just simply an *OR channel* [85] are common as well.

In the wireless world, on-off keying (OOK) has been in use since the beginning of radio. In its simplest form, the transmitter broadcasts wideband noise (such as from a spark-gap generator in the early days) to transmit a mark and silence to transmit a space. The receiver then simply looks for the presence of noise above a certain threshold and calls that a mark while noise below the threshold is called a space. Provided the threshold is not changed as a result of the jammer's actions, the jammer can only convert spaces to marks. In practice however, just using OOK does not yield a viable OR-channel from the jam-resistance point of view because traditional coding schemes are as sensitive to space errors as they are to mark errors. Thus, a jammer normally only has to corrupt a relatively small number of spaces to destroy the information content of the signal. To compensate, the receiver's discrimination threshold is increased even in the presence of mild jamming and it quickly rises to the point that mark-errors as well as space-errors are produced. Like I^2C, jammers can create this situation with very little relative power.

The underlying reason why the OOK receiver must raise its discrimination threshold is because present codecs rely on both the marks and the spaces in the received bitstream to convey information. This is largely forced on them by the desire/need to work with physical channels having symmetric bit-error probabilities and it's consequences carry over to channels that happen to be asymmetric. However, if (1) the channel is highly asymmetric, (2) the receiver does not rely on the presence of any (particular) spaces in the codeword, and (3) they can tolerate a large number of additional marks (space-errors), then the situation changes fundamentally. Under these conditions, the receiver can leave their discriminator threshold set very low and accept the false marks as part of the received packet.

Suitable OOK transmission protocols are possible with each of the three major forms of traditional spread spectrum - direct sequence, frequency hopping, and time hopping. The easiest one to visualize is time hopping spread spectrum such as used

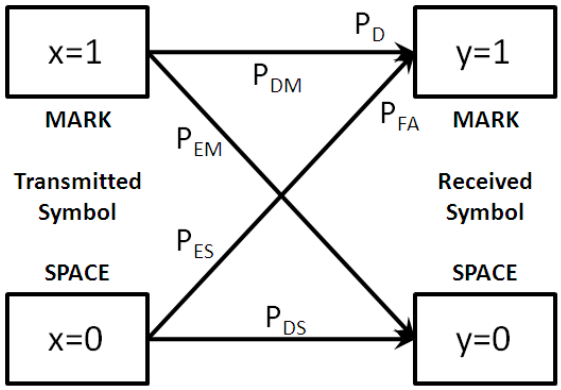

Figure 9.1: Translation paths in a binary channel

in impulse-based ultra-wideband radio [86, 87, 88, 89, 90, 91, 92, 93]. Using this approach, very short but high power pulses are produced at times, relative to the start of the packet, dictated by the codec's encoding algorithm. Pulse durations of fractions of a picosecond are achievable with low cost pulse generators [94].

9.1 Binary Asymmetric Channels

The translation paths from transmitter to receiver in a binary channel are shown in Figure 9.1. The following definitions describe the various probabilities involved:

$$P_{DM} \triangleq P(y = 1|x = 1) = P_D \tag{9.1}$$

$$P_{ES} \triangleq P(y = 1|x = 0) = P_{FA} \tag{9.2}$$

$$P_{EM} \triangleq P(y = 0|x = 1) = (1 - P_D) \tag{9.3}$$

$$P_{DS} \triangleq P(y = 0|x = 0) = (1 - P_{FA}) \tag{9.4}$$

The probability that a mark is detected, given that one was transmitted, is the Probability of Mark Detection, P_{DM}, as defined in (9.1). This is the same as the Probability of Detection, P_D, in the context of a radar system. The probability that

a mark is detected, given that one was not transmitted, is the Probability of a Space Error, P_{ES}, as defined in (9.2). This is the same as the Probability of False Alarm, P_{FA}, in the radar context.

The remaining two paths, namely the Probability of a Mark Error, P_{EM}, and the Probability of Space Detection, P_{DS}, can be expressed in terms P_D and P_{FA} as given in (9.3) and (9.4), respectively. For the remainder of this work, we will favor the terms P_D and P_{FA}.

Note that using the P_D and P_{FA} represents a departure from the treatment generally used in communication systems. The general focus is on the two error rates directly, P_{EM} and P_{ES}. The are typically assumed to be equal and are then referred to simply as the Probability of Error, P_E. However, it must be kept in mind that traditional systems generally take steps, through source coding if necessary, to place roughly equal information in transmission symbol, thus making both the relative frequency of each symbol comparable as well as the cost for reception errors of each type. In concurrent coding, much like radar, on the other hand, virtually all of the information is conveyed in the marks and the penalties for missing a mark are much higher than falsely detecting one.

9.2 Generalized On-Off Keying

A concurrent codec ideally operates over an OR channel in which the received packet represents a ORing of all of the packets broadcast. Such a channel can be approximated if the probability of detection for a mark sufficiently high, even in the face of hostile interference. The need to minimize the likelihood that an attacker can force the receiver to make mark errors, leads to the adoption of a generalized form of On-Off Keying (OOK) as the preferred modulation scheme.

Consider the case of any modulation method in which one waveform is broadcast for a mark and another is broadcast for a space. The receiver then has to determine if the received signal looks more like a mark's waveform or a space's waveform based on evaluating which one is the stronger of the two. This means that the attacker can force mark errors simply by broadcasting sufficiently strong spaces. For instance, if a frequency shift keying (FSK) modulation were adopted then a space would correspond to a signal at one frequency while a mark would map to a signal at a second frequency. The attacker could then cause mark errors by transmitting a signal at the space's frequency. Furthermore, since the attacker would know that they only need to cause a very limited number of mark errors in order to cause the loss of an entire packet, then randomly transmitting very strong spaces at low duty cycles would constitute a devastating attack.

As an example, consider a system in which a codeword consists of one thousand marks in a one million bit codeword. Now consider that the attacker randomly transmits space signals having the same energy at the receiver as the legitimate mark signal has. If a space signals collide with any of the mark signals, then there is a 50/50 chance that the mark will be erased and the entire message lost. In order to have a 50% probability that a space signal will collide with at least one of the one-thousand marks, the attacker only needs to broadcast with a signal consisting of spaces with a density of 0.07%. Since the legitimate signal is a constant envelope waveform, the jammer can therefore expend less than 1/1000 of the energy of the legitimate sender while having causing about 25% of all packets to be lost. Furthermore, this is largely independent of the length of the codeword and therefore none of the spectrum spreading achieved by using sparse codewords provides a benefit.

By only transmitting (and only looking for) the signal for a mark, the attacker is left with only two options - transmitting enough marks to overload the receiver's de-

coder or figure out a way to destructively interfere with a mark's waveform sufficiently so as to reduce it below the detection threshold.

Another argument against a traditional constant-envelope modulation scheme is that, unlike binary symmetric channels, the value of a space being correctly received is almost negligible compared to correctly receiving a mark. Therefore it makes little sense to expend the same amount of energy asserting each. Instead, it makes sense to expend little to no energy on asserting spaces and, instead, use that energy to bolster the all-important marks. With a typical mark density of 0.1%, this means that marks can be asserted with an effective power of 1000W for the total energy expenditure of a 1W constant-envelope scheme.

This is not to say that OOK is without drawbacks. In particular, the difficulty of achieving high peak-to-average ratios (PAR) is well known. However, this is is also a problem that has long faced the radar community and several approaches to addressing it have been developed, including various pulse compression techniques [95]. Using these techniques, wide pulses with lower peak power demands can provide time resolutions comparable to higher peak power pulses having the same total energy.

9.3 Mark/Space Waveform Options

The fact that only the presence of a mark must be detected and that severe distortion in its shape can be tolerated permits, in theory, nearly any waveform to be used as a mark symbol. Conversely, since multiple overlaid spaces should still be seen as a space, and not a horribly distorted mark, the most reasonable choice for the space symbol is silence. Thus, we are steered toward using a unipodal signaling scheme as follows:

$$s_{space} \triangleq 0 \tag{9.5}$$

$$s_{mark} \triangleq \begin{cases} s_m(t) & t_0 \leq t \leq (t_0 + T_S) \\ 0 & \text{otherwise} \end{cases} \tag{9.6}$$

Where t_0 is the start time for the particular mark being transmitted and T_S is the symbol duration.

Perhaps the most promising mark waveform comes from DSSS systems [96]. In this approach, a generalized hierarchical Golay (GHG) sequence, such as that employed as a synchronization code in IMT-2000 W-CDMA cell systems [97], might be used as the mark waveform. The transmitter walks through the codeword and broadcasts the entire sequence each time a mark is encountered. The result is a superposition of identical sequences offset in time by the relative time between codeword mark boundaries. The receiver employs an efficient GHG matched filter to detect marks at those locations where the correlation exceeds the detection threshold.

Another potential mark waveform, derived from FHSS, involves associating codeword bits with a two-dimensional grid of frequency/time bins. The transmitter walks across the codeword along the time axis and broadcasts a signal on every frequency in that time slot whose bin contains a mark. The receiver must be capable of listening to all of the frequencies in the hop set and detecting marks associated with any bin whose signal rises above the threshold. At one extreme, there would be a single time slot and the hop set would have as many frequencies as there are bits in the codeword, which can be on the order of one million; but at the other extreme, the hop set could consist of a single frequency, in which case there would be as many time slots as codeword bits. The latter is simply the pulse-based system, described next, by another name. The size of the hop set would depend on the capabilities of the receiver and, as software-defined radios improve, this technique is likely to become increasingly attractive.

Arguably the simplest concurrent code system to implement is a pulse-based system in which the transmitter walks across the codeword and transmits a short, high-power pulse of RF energy each time it encounters a mark [24]. The shape of the pulse can range from random broadband noise to a single-tone carrier. The former is very similar to impulse-based Ultra Wide Band (UWB) systems while the latter is nothing more than the continuous wave (CW) on-off keying (OOK) that dates back more than a century.

One major disadvantage of the basic pulse-based approach is that if the duration of the pulse is limited to the codeword bit period, then transmitters need to be capable of very high instantaneous power output to achieve even moderate data rates along with high degrees of jam resistance. As mentioned previously, this same problem has long faced radar system designers and the fruits of their efforts can be applied here in the form of pulse compression techniques. One such option would be to use a linear frequency modulated (LFM) chirp for each mark. The chirp duration could span many mark periods while still permitting the receiver to discriminate the presence of marks at the necessary temporal resolution. By spreading the mark pulse out in a relatively long chirp, the transmitter can put the same total energy into the mark waveform while significantly reducing the peak power required to do so. In fact, the DSSS-based approach using Golay sequences is an example of phase-coded pulse compression.

9.4 Detection Options

As long as transmitted marks do not intentionally overlap, meaning that the mark waveform duration does not exceed the codeword bit rate, then a simple radiometer offers a suitable detector. In fact, if the details of the mark waveform are not deterministic, as would be the case with pulses of random noise, then a radiometer is

the optimal detector [83, 23]. If the mark waveform is deterministic, then a matched filter detector is the best choice [83, 98]. If the symbol duration is longer than the codeword bit period, then there is no choice but to use a deterministic signal since the receiver must correlate the received waveform against the expected mark waveform to obtain a mark location in time.

9.5 Pulse-Based Receiver Performance

For the remainder of this dissertation, attention will be focussed on a pulse-based system using an arbitrary, but deterministic, mark waveform that is bounded by the codeword bit period. Not only is this arguably the most basic system with which to perform the initial comparison to existing spread spectrum systems, but it is most representative of systems that have been implemented and field tested in jamming environments [99, 100].

9.5.1 Receiver Architecture

To date, four signal detection approaches have been envisioned for CCSS. These are a correlator (or, equivalently, a matched filter), a radiometer, an envelope detector, and an FFT-based approach. At present, only software-defined implementations of the first two have been used in actual practice. This dissertation will analyze only the matched filter detector. Consequently, the receiver architecture under consideration is that shown in Figure 9.5.1.

The signal $x(t)$, at the input to the matched filter, is the sum of the information signal, $s(t)$, and the noise signal, $n(t)$. The noise signal is composed of two components: nonmalicious noise that is that normally present in the channel (particularly the noise associated with the receiver front end) and malicious noise broadcast by other transmitters (in particular, a jammer). Nonmalicious noise will be assumed to

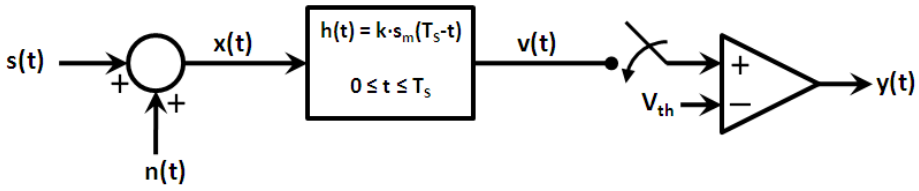

Figure 9.2: Receiver architecture.

be additive white Gaussian noise (AWGN) that may be bandlimited as appropriate. The malicious noise may or may not be AWGN as dictated by the type of attack being considered. The signal at the filter output, $v(t)$, is sampled at the end of the symbol period and the sampled signal presented to the comparator. The output of the comparator is a 1 (mark) if its input is above the threshold value, V_{th}, and a 0 (space), if below.

The information signal, $s(t)$, is assumed to either be that for a mark, $s_m(t)$, or that for a space, $s_s(t)$. For convenience, and without loss of generality, it is assumed that $s(t)$ is defined on the interval between zero and the symbol duration, T. Furthermore, since the transmitter is squelched when spaces are transmitted, $s_s(t)$ is identically zero.

The noise signal, $n(t)$, is assumed to be additive white Gaussian with single-sided noise power spectral density N_0. Thus, the noise variance is

$$\sigma_n^2 = \frac{N_0}{2} \tag{9.7}$$

As is common practice with this type of analysis, certain simplifying assumptions will be made. In particular, the matched filter is assumed to incorporate a rectangular bandpass filter that just encompasses the signal bandwidth. Furthermore, it will be assumed that the receiver is perfectly aligned with arriving symbols. While overly optimistic, these form a basis of comparison with the comparable analyses for traditional modulation schemes.

9.5.2 Receiver Analysis

For the receiver described in the previous section, we are interested in the voltage distribution on the sample-and-hold as a function of the symbol transmitted and the background noise. To determine this distribution, we will walk through the receiver and determine the distribution at each step.

The signal at the input of the filter is

$$x(t) = s(t) + n(t) \tag{9.8}$$

where, as described previously, $s(t)$ is identically zero when a space is transmitted and equal to a pulse function, $s_m(t)$ that, for this analysis, is non-zero only within the duration of one symbol period. The noise, $n(t)$, is AWGN with single-sided power spectral density of N_0.

The output of the filter is

$$v(t) = (s(t) + n(t)) * h(t) \tag{9.9}$$

For a binary channel in AWGN, the matched filter provides the best discrimination between two deterministic symbol waveforms [98]. The impulse response, $h(t)$, of such a filter is the time-reversed version of the difference between the two symbol waveforms. In the case of unipodal signalling, which one of the symbol waveforms is identically zero, the matched filter impulse response becomes simply a time-reversed version of the non-zero symbol waveform, namely

$$h(t) = ks_m(T_S - t) \tag{9.10}$$

where k is an arbitrary scaling constant. In general, k is not dimensionless; thus, while k is arbitrary, we will refrain from assigning it a particular value and allow it to cancel naturally.

This signal at the output of the filter is the convolution of the input signal, $x(t)$, and the filter's impulse response, $h(t)$.

$$v(t) = (s(t) + n(t)) * ks_m(T_S - t) \tag{9.11}$$

Since the filter is linear, this can be expressed as the sum of a symbol component and a noise component.

$$v(t) = v_s(t) + v_n(t) \tag{9.12}$$

$$v_s(t) = s(t) * ks_m(T_S - t) \tag{9.13}$$

$$v_n(t) = n(t) * ks_m(T_S - t) \tag{9.14}$$

The symbol component is deterministic and, when the symbol is a space, is identically zero. When a mark is present, the filter output at the sampling instant, $t = T_S$, is

$$\begin{aligned} v_m(T_S) &= s_m(T_S) * ks_m(T_S - t) \\ &= \int_{-\infty}^{+\infty} s_m(t) ks_m(t) dt \\ &= k \int_{-\infty}^{+\infty} s_m^2 dt \\ &= kE_m \end{aligned} \tag{9.15}$$

where

$$E_m = \int_{-\infty}^{+\infty} s_m^2(t) dt \tag{9.16}$$

is the normalized energy in the mark.

Turning to the noise component, because the filter is linear and the input is Gaussian, the output is also Gaussian with a scaled mean and variance. Additionally, because the input noise is zero mean, the output is also zero mean. The variance of the output can be determined by multiplying, in the frequency domain, the input noise power spectral density and the frequency response of the filter.

$$\sigma_s^2 = \int_{-\infty}^{+\infty} \frac{N_0}{2} \left|H(f)\right|^2 df \tag{9.17}$$

Using Parsevals theorem, the variance can then be written as

$$\begin{aligned} \sigma_v^2 &= \int_{-\infty}^{+\infty} \frac{N_0}{2} |H(f)|^2 df \\ &= \frac{N_0}{2} k^2 \int_{-\infty}^{+\infty} s_m^2(t) dt \\ &= \frac{N_0}{2} k^2 E_m \end{aligned} \tag{9.18}$$

The net result is that the pdfs at the comparator input can be expressed by

$$f_i(v) = \frac{1}{\sqrt{2\pi\sigma_v^2}} e^{-\frac{(v-\mu_i)^2}{2\sigma_v^2}} \tag{9.19}$$

where $i = s$ for a space and $i = m$ for a mark and

$$\sigma_v^2 = \frac{N_0}{2} k^2 E_m \tag{9.20}$$

$$\mu_s = 0 \tag{9.21}$$

$$\mu_m = k E_m \tag{9.22}$$

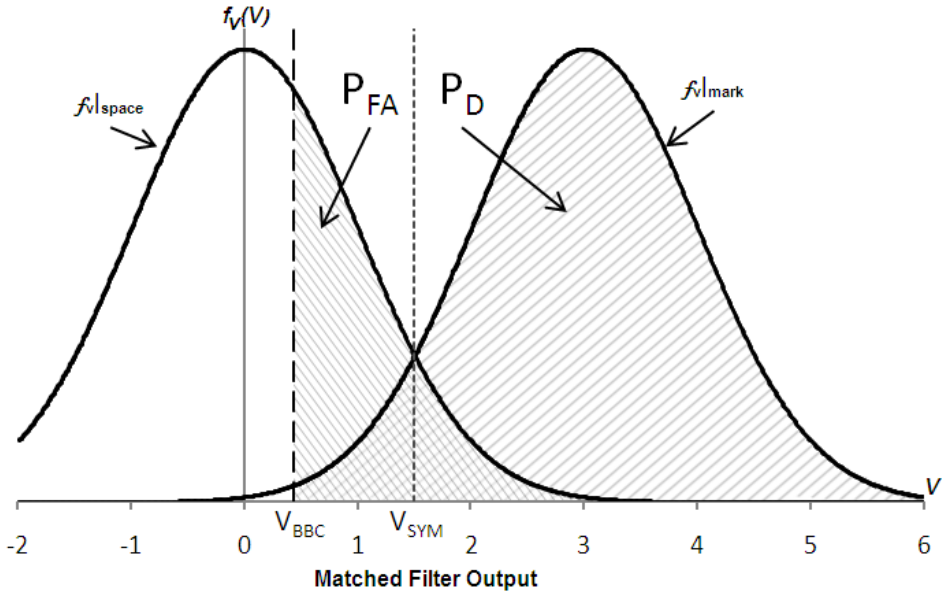

Figure 9.3: Threshold selection.

9.5.3 ROC for Pulse-based CCSS

Since the pdf of the signal at the comparator input, for this receiver, is zero-mean Gaussian for a space and non-zero mean Gaussian for a mark, both having the same variance, the distributions are typified by Figure 9.5.3. In a symmetric channel, the lowest overall bit error rate is achieved when the threshold is placed at the intersection of the two curves, indicated by the dotted line, making the individual bit error probabilities equal. In an asymmetric channel, however, the optimal threshold selection is usually different and should be determined via hypothesis testing. The details of this will be covered in the next section, but for a system based on Standard BBC, the optimal threshold is indicated by the dashed line.

The quantitative analysis of this system begins with mapping out the Receiver Operating Characteristic (ROC), which is a plot of the Probability of Detection, P_D, versus the Probability of False Alarm, P_{FA}. We will develop these relations parametrically.

When no mark is present, the detector will produce a false alarm if the threshold, V_{th}, is set low enough such that signal at the comparator input exceeds it. The probability of this occurring is

$$P_{FA} = P(v > V_{th}|\text{space}) = \int_{V_{th}}^{+\infty} f_s(v)dv \tag{9.23}$$

Similarly, when a mark is present, the detector will succeed in detecting it if the combined signal and noise is above the comparator threshold. The probability of this occurring is

$$P_D = P(v > V_{th}|\text{mark}) = \int_{V_{th}}^{+\infty} f_m(v)dv \tag{9.24}$$

These results can be expressed in terms of the Q-function [83], which is defined as

$$Q(x) \triangleq \int_{x}^{+\infty} Z(u)du; \qquad x \geq 0 \tag{9.25}$$

where, in turn, $Z(x)$ is the normalized Gaussian pdf defined as

$$Z(x) \triangleq \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{9.26}$$

Because $Z(x)$, and therefore $Q(x)$, are normalized about the mean, positive values of x represent values above the mean. Applying (9.25) and (9.26) to (9.23) and (9.24), the ROC can be expressed parametrically as

$$P_D = Q\left(\frac{V_{th} - \mu_m}{\sigma}\right) \tag{9.27}$$

and

$$P_{FA} = Q\left(\frac{V_{th}}{\sigma}\right) \tag{9.28}$$

Our final step, in this section, is to express the threshold voltage in a more meaningful way. An obvious choice is to express it as a fraction of the expected filter output

when a mark is present, which is μ_m. Since the filter output voltage represents signal energy, this is analogous to expressing the decision threshold in terms of the nominal mark energy.

$$\begin{aligned} V_{th} &= \alpha \mu_m \\ &= \alpha k E_m \end{aligned} \tag{9.29}$$

At the same time, the SNR-per-bit, ξ, is given by

$$\xi \triangleq \frac{E_b}{N_0} \tag{9.30}$$

which, combined with (7.13), yields

$$\frac{E_m}{N_0} = \eta \xi \tag{9.31}$$

These, in combination with the relations in (9.20), allows us to express the parametric curves compactly as

$$P_D = Q\big((\alpha - 1)\sqrt{2\eta\xi}\,\big) \tag{9.32}$$

and

$$P_{FA} = Q\big(\alpha\sqrt{2\eta\xi}\,\big) \tag{9.33}$$

Note that, during this process, the arbitrary scaling constant, k, in the matched filter cancels out entirely.

Figure 9.5.3 shows the ROC plots for several values of mark SNR that span the region having probabilities of detection between 99.99% and 99.999% and false alarm rates between 10% and 50%. This is a region of practical interest when considering

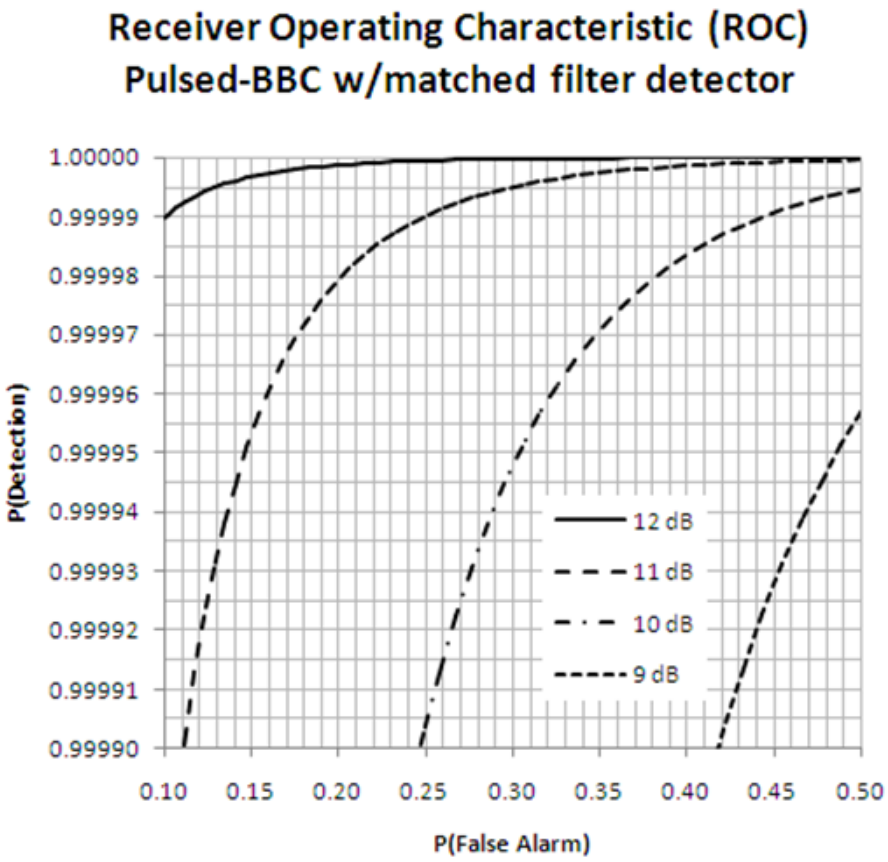

Figure 9.4: Receiver Operating Characteristic for region of practical interest for various values of SNR/bit.

expected BER performance of a pulsed-BBC system based on this receiver architecture.

9.5.4 Threshold Selection

Using the ROC curves from the previous section, we can quantify the expected performance as a function of both the threshold voltage and the SNR-per-bit, but we must still decide how the actual threshold voltage will be chosen. One option is to use the Bayes Criterion, but this requires not only knowledge of the a priori probability with which each symbol is transmitted, but also knowledge of the costs associated with making incorrect decisions. In fact, this criterion, applied to a channel that is

symmetric both in terms of probabilities and costs, leads directly to the choice of the threshold that equalizes the individual error rates. In an asymmetric channel, however, even when the symbol probabilities are known, the cost functions can't be assumed to be symmetric and are typically hard to determine sufficiently well. Furthermore, in a CCSS system operating in a hostile environment, the attacker exerts dominant control over the actual symbol probabilities seen by the receiver – although that is not an issue in this particular analysis since only AWGN barrage jamming is considered. Fortunately, another means of setting the threshold is available that does not depend on a priori knowledge of the symbol probabilities, namely the Neyman-Pearson Criterion.

In simplified terms applicable to this problem, the Neyman-Pearson Criterion aims to achieve an optimal balance between the Probability of Detection, P_D, and the Probability of False Alarm, P_{FA}. Normally, this would still involve a cost function to quantify the tradeoff between fewer missed legitimate marks and having to deal with more false marks. However, in this case, the cost of missing a legitimate mark is so high and the marginal cost of a false mark so low, up to a limit, that we can express the optimal operating point in very simple terms: Namely, the P_D should be made as high as possible by lowering the threshold until the highest tolerable P_{FA} is achieved. In theory, we could set the P_{FA} such that the decoder operated at or near the critical mark density. However, this would require a decoder capable of processing an arbitrarily large number of messages. Instead, we will choose to operate the decoder at the limit of the threat model, which is a received mark density of 33%.

Because the legitimate mark density, particularly in a jamming environment, is very low, we can achieve a received mark density of slightly, but acceptably, over 33% by setting the threshold so that the P_{FA} is 33%. This is achieved by setting the threshold at

$$V_{opt} = 0.4307\sqrt{\sigma_v} \tag{9.34}$$

This is the optimal threshold depicted in Figure 9.5.3 by the dashed line. As is quite evident, being able to lower the threshold in this way significantly improves the P_D compared to the symmetric channel's optimal threshold. Furthermore, as the SNR-per-bit increases, the advantage becomes more pronounced because the threshold can remain the same, while the symmetric threshold must increase proportionately to keep the two error probabilities in balance.

Actually operating a receiver at the target threshold can readily be accomplished two ways. First, the threshold can simply be adjusted using some form of automatic gain control (AGC) so that the desired mark density is produced, on average. However, this approach offers a hook to the adversary who would strive to find low-energy ways to pull the threshold up enough to produce mark errors. Another approach would be to use a running statistic threshold [80], which would set the threshold for each packet to precisely the level required to produce a mark density, for that packet, that was as close to the target density as possible without exceeding it.

Chapter 10

Performance of Pulsed-BBC CCSS

In this chapter we will take the Receiver Operating Characteristic, ROC, developed in the previous chapter and use it to first develop a family of curves showing the effective bit error rate, BER, as a function of the signal-to-noise ratio on a per data bit basis (SNR-per-bit). To do this, we must first understand how to equate the packet loss rate (PLR) performance in an erasure channel, such as a CCSS system, to an effective BER applicable to an error channel, such as used by traditional spread spectrum systems.

With that done, we will be in a position to generate curves describing how our simple CCSS channel performs in the presence of AWGN barrage jamming. From there, it is a simply matter of overlaying these curves onto similar curves for comparable traditional spread spectrum systems.

Before beginning, we must acknowledge that, in the end, we will still be left with a somewhat apples-to-oranges comparison between systems. This is largely due to two considerations. First, as already noted, CCSS is intrinsically an erasure channel whereas the channels we are comparing it to are error channels. In typical networks, error channels are used to construct the lowest layers, such as the physical and data link layers. Somewhere around the network and transport layers, protocols are used

to turn the overall link into an unreliable erasure channel in which additional overhead information is added to the data stream and protocols, such as any of the variety of common automatic repeat request (ARQ) protocols are used to deal with lost packets, either through erasure coding, retransmissions, or a combination of the two. Since CCSS is already an erasure channel, it is much closer to being at the transport layer than most traditional physical layer transmission schemes. As a consequence, in order to compare the performance of the two directly, the comparison should be made at a level in the network stack where, from the perspective of the higher layers, the two can be treated as interchangeable black boxes. Such a comparison is beyond the scope of this dissertation and therefore we are left with using a fairly simple comparison based on the information theoretic channel capacities of the two and ignoring the impact of any additional overhead associated with the steps needed to make them truly equivalent.

The second consideration that makes direct comparison problematic, particularly for the simple pulse-based system analyzed here, is that traditional spread spectrum systems almost always use constant-envelope modulation schemes that transmit at a constant power output level (in terms of the signal envelope, which is a measure of the mean power output over a relatively short time period, such as one symbol or even one cycle of the carrier). The pulse-based CCSS system analyzed here has a very high peak-to-average ratio (PAR); for the typical system we have been using as our example, the ratio is roughly 1000:1. This poses a number of practical considerations, many of which were discussed in Chapter 9. Consequently, it could be argued that comparing a CCSS system to a traditional system at the same mean power output gives an unfair advantage to the CCSS system since it would be likely to be larger, heavier, and more expensive to produce. However, comparing them on the basis of peak power would similarly tilt the tables in favor of the traditional systems, since, for example, an amplifier that can generate 100W of envelope power continuously

would be larger, heavier, and more expensive than one that can produce 100W of peak power but is only called on to sustain 100mW of mean power. Therefore, we choose to ignore these considerations and perform the comparison based on mean power for the following reasons:

1. There is no reasonable way to determine where, in between the two extremes, the comparison should be made since it will be strongly dependent on the specific systems and applications involved.

2. Any comparison point chosen today would likely be unsuitable in the future as technological developments push the relative costs in favor of one or the other in largely unpredictable ways.

3. The analyses of traditional systems, at this level, almost always ignore practical considerations in the actual analyses, although they may be mentioned alongside (as we are doing here).

4. The analyses of traditional systems generally use mean power as the comparison point between the legitimate and hostile transmitters, even when considering hostile waveforms that possess high, even unrealistic, PAR values.

In the end, our analysis is therefore very similar to comparable analyses of traditional systems found in the literature; namely, we present a theoretical analysis that ignores practical constraints and leaves it up to the users of that analysis to apply appropriate bounds based on the constraints applicable to them.

10.1 Comparing Erasure and Error Channels

A BBC-based concurrent code system is intrinsically atomic at the message level, meaning that either an entire message is received, intact and error free, or the entire

message is lost. This makes the system an example of an erasure channel, as opposed to the more traditional error channel. Since, presumably, each message contains overhead information, such as sequence numbers, the system could employ any of the conventional ARQ (automatic repeat request) schemes in use today to identify and retransmit lost messages. Furthermore, like most conventional systems, suitable forms of FEC (forward error correction) coding could be used to recover lost messages based on the content of the received messages. In general, erasure codes are more efficient than the more general error correcting codes because error codes know which information was lost. For instance, the use of a single parity bit can detect all single-bit errors, but has no ability to correct them. But in an erasure channel, a single parity bit can be used to reconstruct any single-bit losses.

While the analysis in this dissertation ignores the potential impact of any error correction applied to any of the systems compared, the fact that CCSS uses an erasure channel and that conventional systems use error channels requires that the fundamental differences be taken into account. This is because error channels characterized by an expected bit-error-rate (BER), whereas erasure channels are characterized by an expected bit-loss-rate (BLR). Simply comparing the BER to the BLR does not form a valid comparison of the two. However, it is possible to convert a particular BLR into an information theoretic equivalent BER and thus provide a valid means of comparison.

In an error channel, each received symbol has some probability of containing an error. However, in the absence of additional forward error correction (FEC) coding, there is no way to distinguish which symbols do and which do not have errors. Thus, as the performance worsens, the BER approaches 50%. At this point, half of the received bits are actually correct, but there is no way to know which half; thus no information is conveyed at all.

In an erasure channel, each symbol is either received correctly or not received at all. In the case of BBC, the scope at which this occurs is the message, not the mark. In Standard BBC, if the receiver makes a mark error, then any message for any codeword containing that mark will be lost. Conversely, however, any message that the decoder successfully extracts from the signal has only a vanishingly small probability of containing any errors at all. Thus, if the BLR is 50%, then the half of the bits that are received still correctly convey useful information.

Since all of the information is obviously being conveyed when either the BER or BLR is 0%, these points are equivalent. Similarly, since complete loss of information conveyance occurs when the BER is 50% or the BLR is 100%, these points are equivalent. However, beyond being monotonic, the shape of the curve between these endpoints is not obvious. For that, we can draw upon the field of information theory and note that each channel, for a given error rate, has a particular channel capacity. We can then consider the performance of two channels equivalent if their channel capacities are equal. Thus, once a BLR is computed for a CCSS channel under a given set of conditions, it can be described as having the equivalent BER of an error channel having the same capacity.

For a binary symmetric channel with a bit error rate BER, the channel capacity [9] is

$$C_{\mathrm{BSC}} = 1 - H(\mathrm{BER}) \tag{10.1}$$

where $H(p)$ is the entropy function

$$H(p) = -p\ \log_2(p) - (1-p)\ \log_2(1-p) \tag{10.2}$$

For a binary erasure channel with bit loss rate BLR, the channel capacity is

$$C_{\mathrm{BEC}} = 1 - \mathrm{BLR} \tag{10.3}$$

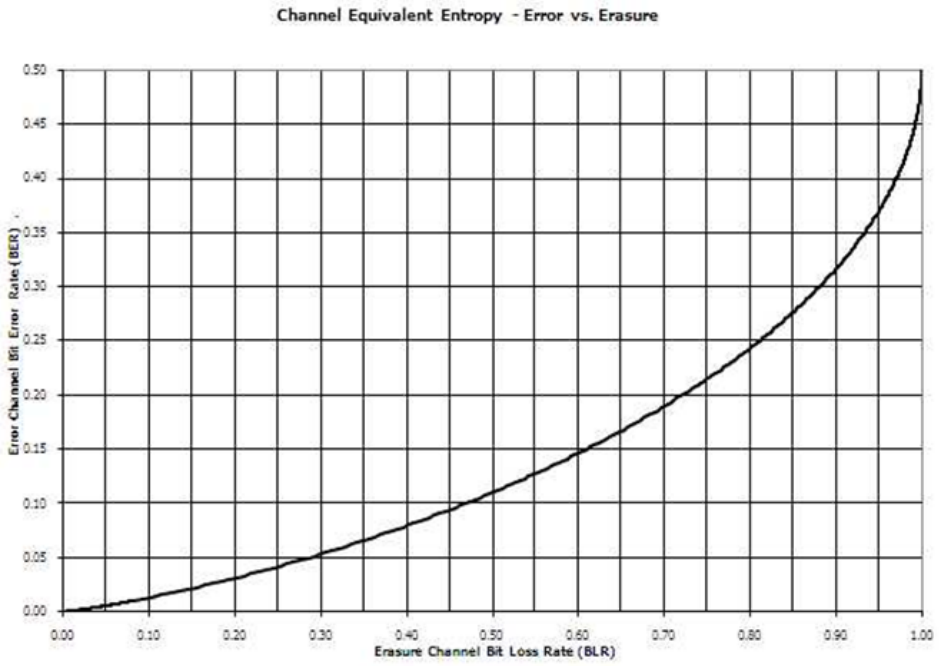

Figure 10.1: Equivalence relation between binary erasure channels and binary error channels.

Setting the two expressions for channel capacity equal and solving for one error rate in terms of the other, the conversion from BEC error rate to an equivalent BSC error rate becomes

$$\mathrm{BLR} = H(\mathrm{BER}) \tag{10.4}$$

Therefore, the inverse entropy function can be used to convert from a BBC bit loss rate to an equivalent bit error rate for a traditional system implemented using a binary symmetric channel. The mapping between the two channels is shown in Figure 10.1.

Since we are seldom interested in BERs of more than a few percent, we can take note from the mapping that BLR values below about 10% correspond to BER values approximately an order of magnitude less. Corresponding values for selected BET values are shown in Table 10.1. Thus, for example, to achieve a BER of 1%, we can tolerate a BLR of slightly over 8%.

Table 10.1: Selected BLR vs. BER pairs.

BER	BLR
50.0%	100.0%
33.3%	91.8%
10.0%	46.9%
1.0%	8.1%
10^{-3}	1.1×10^{-2}
10^{-4}	1.5×10^{-3}
10^{-5}	1.8×10^{-4}
10^{-6}	2.1×10^{-5}
10^{-7}	2.5×10^{-6}

To appreciate the impact of using an error channel, consider its effect on our stock example, namely a Standard BBC codec using $L = 1000$-bit messages and $K = 30$-bit checksums. Including the pair of bookend marks, each codeword will nominally have 1,032 marks. If we fail to detect even a single mark, the entire packet will be lost. The probability that we will detect all 1,032 is simply the product of the probability of detecting each mark individually. Thus, the likelihood that we will codeword will be erases, which also becomes the BLR, is

$$\text{BLR} = \text{PLR} = 1 - P_D^{L+K} \tag{10.5}$$

Solving this for a 1% BLR, we get a required P_D of 0.9999903 (or a mark loss rate, MLR, of 1.1×10^{-5}). However, to achieve a BER of 1%, we only need to achieve a BLR of 8.1% and, to achieve that, we can tolerate a noticeably higher MLR of 8.2×10^{-5}.

10.2 Bit Error Rate Performance

The final step in the analysis is to combine the parametric equations so as to produce a plot of the BER as a function of the SNR-per-bit. The assumption in this analysis

is that the transmitter is configured to place a single message in each packet and to transmit them in a non-overlapping fashion. Under these conditions, each mark error places only a single message at risk. If these assumptions are not valid, then a single mark error could place multiple messages at risk, but since the transmitter has control over this, it is a reasonable assumption to make. For the basic BBC algorithm, if a single mark associated with a message is lost, the entire message is lost. It should be noted that more advanced forms of the BBC algorithm, particularly block-oriented BBC, encompass coding enhancements that permit a limited number of mark errors to occur before a message is lost. However, this analysis will remain limited to the basic algorithm. Thus, the probability of a message being successfully received is

$$P_{msg} = P_D^N \tag{10.6}$$

where N is the nominal number of marks in a message. The resulting bit loss rate is therefore

$$\text{BLR} = \text{MLR} = 1 - P_{msg} = 1 - P_D^N \tag{10.7}$$

To simplify things, we will assume that the detector threshold is always between 0 and E_m. Note that this assumption is not necessarily valid in enhanced versions of BBC that permit the false alarm rate to be arbitrarily high, thus resulting in negative threshold values. For $0 \leq \alpha \leq 1$

$$P_{FA} = Q(\alpha\sqrt{2\eta\xi}) \tag{10.8}$$

and

$$P_D = Q\Big((1-\alpha)\sqrt{2\eta\xi}\Big) \tag{10.9}$$

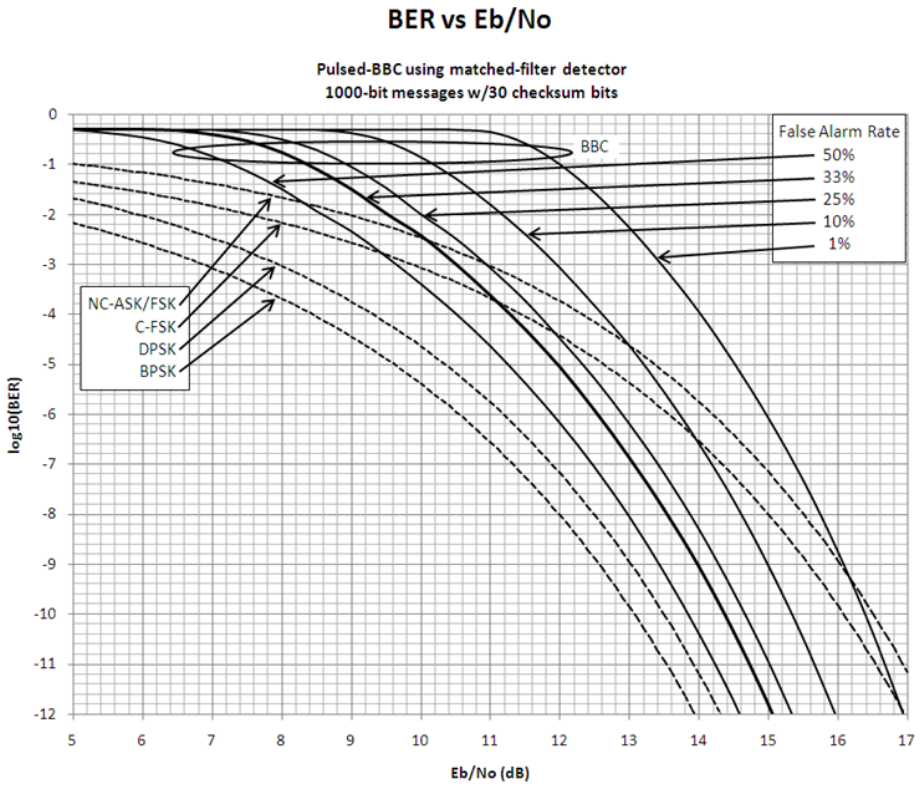

Figure 10.2: BER for pulsed-BBC w/matched filter detector in AWGN.

Thus, the BER, taking into account the mapping from BLR to equivalent BER, as a function of the SNR-per-bit and the desired false alarm rate is

$$\text{BER} = H^{-1}\left(1 - \left[1 - Q\left(\sqrt{2\eta\xi} - Q^{-1}(P_{FA})\right)\right]^{N}\right) \tag{10.10}$$

Note, in particular, that the SNR-per-bit is on a per-data-bit basis, and not on a per-mark basis. The impact of the additional marks associated with checksum bits or enhanced codec schemes, such as multimark BBC, are accounted for by means of the concurrency efficiency parameter, η.

The BER curves for 1000-bit pulsed-BBC messages with 30 checksum bits are shown in Figure 10.2 for several false alarm rates. The primary curve of interest is the 33% rate since this is the edge of the BBC threat model.

Because of the large message size (typically about 1000 bits) and the all-or-nothing nature of BBC at the message level, the underlying mark error rate must be well below one part in the message length before any appreciable number of messages start being received at all; this is an intrinsic artifact of BBC. Also shown are the BER curves for several traditional modulation schemes [83]. Because BBC is intrinsically all-or-nothing at the message level, the mark loss rate must drop well below one lost mark per number of bits in a message before meaningful numbers of messages are successfully recovered. However, once this occurs, the performance improves much more rapidly than for traditional symmetric channels because, as the SNR-per-bit improves, the discrimination threshold can remain fixed relative to the noise floor instead of needing to rise in order to reduce the space error rate on pace with the shrinking mark error rate.

10.3 Comparison of CCSS to FHSS and DSSS

Given the performance of pulsed-BBC in AWGN, it is a straight-forward exercise to determine the expected performance in the presence of barrage jamming in which an adversary broadcasts AWGN that has been bandlimited so as to match the spread-signal bandwidth. This can then be compared to the expected performance of existing systems.

Because the jamming noise is uncorrelated to the receivers noise floor, the total double-sided spectral noise density, within the signal bandwidth, becomes

$$\frac{N_T}{2} = \frac{N_0}{2} + \frac{N_J}{2} \tag{10.11}$$

Substituting this into the equations above for P_FA and P_D and then rederiving the equation for the BER yields

$$\text{BER} = H^{-1}\left(1 - \left[1 - Q\left(\sqrt{2\eta\frac{\frac{E_b}{N_0}}{1+\frac{N_J}{N_0}} - Q^{-1}(P_{FA})}\quad\right)\right]^N\right) \tag{10.12}$$

The mean power in the jamming signal is

$$J = N_J W \tag{10.13}$$

where W is the bandwidth of the legitimate signal. Similarly, the mean power in the legitimate signal is

$$P = E_B R \tag{10.14}$$

where R is the data rate. With these relations, the ratio of the spectral densities of the jamming noise and the receiver noise can be expressed as

$$\begin{aligned}\frac{N_J}{N_0} &= \frac{\frac{E_B}{N0}}{\frac{E_B}{N_J}} \\ &= \frac{\frac{E_B}{N0}}{\frac{PW}{JR}}\end{aligned} \tag{10.15}$$

The net result is the following expected performance of pulsed-BBC, using a matched filter detector, in barrage jamming.

$$\text{BER} = H^{-1}\left(1 - \left[1 - Q\left(\sqrt{2\eta\frac{\frac{E_b}{N_0}}{\left(1+\frac{\frac{E_b}{N0}}{\frac{PW}{JR}}\right)} - Q^{-1}\left(P_{FA}\right)}\quad\right)\right]^N\right) \tag{10.16}$$

The resulting curves are shown in Figure 10.3 for several different values of SNR-per-bit (and, again, this is on a per-data-bit basis). The 30dB curve is essentially at the asymptotic limit (for the range of BER values shown).

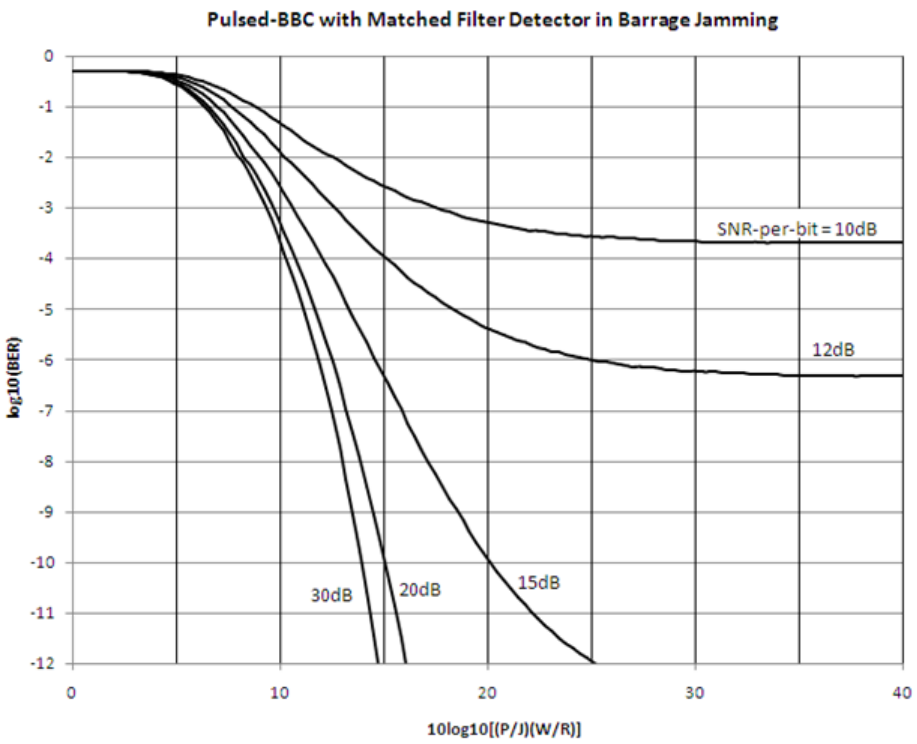

Figure 10.3: Pulsed-BBC in AWGN Barrage Jamming.

The independent variable, $(P/J)(W/R)$, in Figure 10.3 can be viewed from several perspectives. Most fundamentally, it is the ratio of the energy per bit in the signal to the mean noise power spectral density of the jammers signal, namely E_b/N_J. Thus each curve traces out the BER performance of the system, at a given SNR-per-bit relative to the non-hostile noise, as the energy-per-bit in the legitimate signal increases relative to the jammer's overall power spectral density over the channel bandwidth. However, this hides that fact that it also implicitly normalizes out the processing gain, in terms of jam resistance due to the spreading, since if the signal bandwidth, W, increases then the average jammer power, J, must increase by the same factor to stay at the same point on the plot and, hence, force the same BER. Another way to view the axis is as a cost-per-benefit index where the cost is the bandwidth required per data rate achieved (W/R), divided by the benefit, which is the amount of mean

power the jammer must commit per unit of mean power employed by the legitimate sender (J/P).

To compare the performance, against barrage jamming, of pulsed-BBC to traditional forms of spread spectrum, we can plot the performance curves of all three on a common set of axes. The performance of DS(BPSK)/SS in barrage jamming [83] is

$$BER_{DS} = Q\left(\sqrt{\frac{2}{K\left[\left(\frac{N_0 R}{P}\right) + \left(\frac{J}{P}\right)\left(\frac{R}{W}\right)\right]}}\right) \tag{10.17}$$

where K is a measure of the reduction in jamming signal power due to the pre-despreading filter. While this value might be as low as 0.903 for BPSK direct sequence, we will be conservative and assume it to be equal to unity, which it quickly approaches if the pre-despreading filter is much larger than the direct signal bandwidth or if minimum shift keying, MSK, is used as the underlying modulation scheme.

The performance of FH(BFSK)/SS, again in barrage jamming [83], is

$$\mathrm{BER_{FH}} = \frac{1}{2} e^{-\frac{1}{2}\frac{1}{\left(\frac{N_0 R}{P}\right)+\left(\frac{J}{P}\right)\left(\frac{R}{W}\right)}} \tag{10.18}$$

Plotting the curves is one thing, but meaningful comparisons require consideration of what the range of reasonable operating points on the $(P/J)(W/R)$ axis is, as well as what are reasonable values for the SNR-per-bit. The strict answer is, of course, that it depends on the specific situation and application. Having stated that, however, it can be observed that a commonly accepted rule-of-thumb is that SNR-per-bit values below about 10dB are only marginally useful whereas values in the range of 15dB are frequently quite usable, though significantly higher values are often needed, particularly in fading channels (which are not analyzed here). This rule-of-thumb appears to have originated in the early days of radar but also appears to still be useful for modern communication systems, including DSSS and FHSS. The degree to which it is applicable to a concurrent channel is not yet known, but it is at least

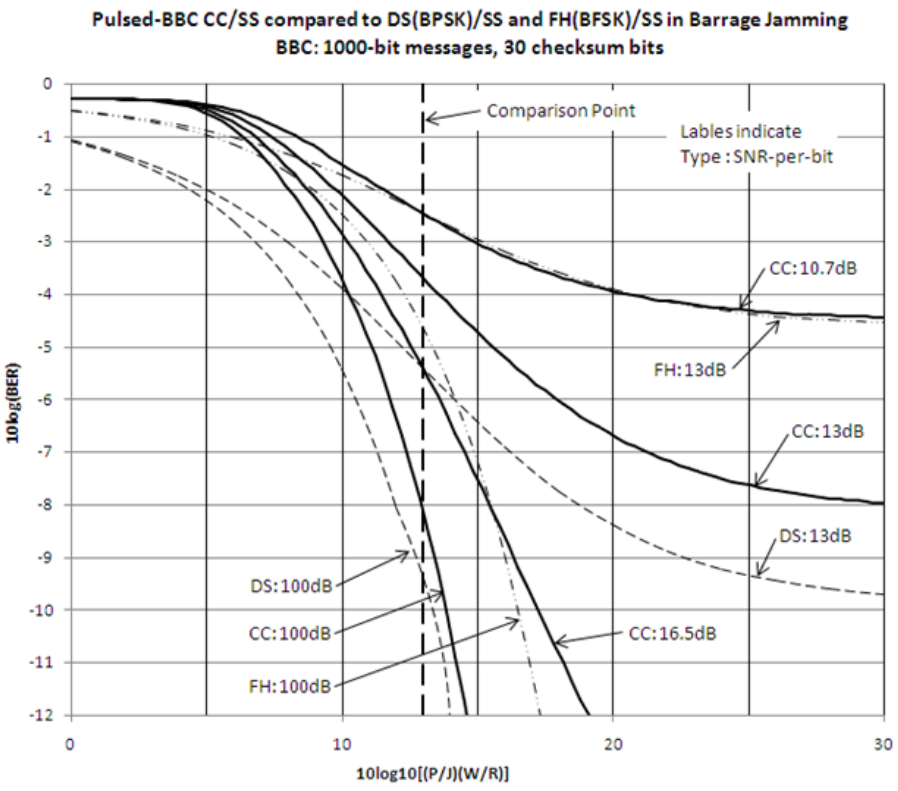

Figure 10.4: Comparison of CCSS to DSSS and FHSS in barrage jamming.

a starting point. Turning to the $(P/J)(W/R)$ axis, we note that if E_b/N_J is equal to E_b/N_0, then the total SNR-per-bit is reduced by 3dB relative to the SNR-per-bit in thermal noise alone. Thus, if we choose 13dB as the SNRs involved, the total SNR-per-bit will be 10dB, which is the floor of our rule of thumb. The resulting comparison plot is shown in Figure 10.3

Two curves each are plotted for all three systems, namely SNR-per-bit values of 13dB, the floor of the range of interest, and 100dB, which is essentially saying that noise in the receiver is negligible and that all performance degradation is due to the jamming signal. For $(P/J)(W/R)$, or equivalently, Eb/NJ, values above approximately 8dB-per-bit, the performance of CCSS is between that of FHSS and DSSS. In addition to the above curves, two additional curves are plotted for CCSS. The first, with an SNR-per-bit of 16.5dB, achieves the same expected performance as the 13dB-per-bit direct sequence system at the $(P/J)(W/R)$ comparison point of 13dB. This

indicates that pulsed-BBC underperforms DSSS by about 3.5dB (at this point). The other curve, with an SNR-per-bit of 10.7dB, achieves the same expected performance as 13dB-per-bit frequency hop system at the same comparison point, indicating that pulsed-BBC actually outperforms FHSS (again, at this point).

Chapter 11

Demonstrators

11.1 Visualization Demonstrator

To better communicate the impact and degree of jam resistance afforded by concurrent codes a visual demonstration was developed that represents packet data as a Windows bitmap (BMP) file. The image is created by walking along the scan lines (which, for BMP files, begin in the lower left corner) and encoding each pixel white for a space and black for a mark. The program itself is ANSI-C compliant and, hence, cannot render the BMP file directly. Instead, once the BMP file is generated it may be opened in any of the readily available image viewers/editors (such as Windows Paint). In addition to viewing the packet data to gain an appreciation for how sparse and random the packet really is, the user can also modify the image to add additional marks. Once finished, they can use the program to retrieve the packet from the BMP file, decode it, and display any messages found.

The demonstration has proven very effective in conveying the concept and power of concurrent codes to numerous audiences consisting of people from both technical and non-technical backgrounds. In addition, a decoding log capability was added that permits graphing the number of calls to the hash function made at each step of the

decoding process. This acts as a good measure of the amount of effort performed by the receiver. Comparing this to the amount of effort predicted by the theory shows them to be in remarkably close agreement, as already shown in Figures 7.1.5 and 7.2.1.

To further improve the utility of the demonstration, Dr. Dennis Schweitzer[1] developed a Java-applet [101] that permits the User to perform all of these tasks using a single GUI. In addition, Dr. Schweitzer added a graphical representation of the decoding tree that clearly reveals the amount of additional work performed by the receiver as a function of the amount of corruption added by the attacker. The addition of this visualization element resulted in the somewhat startling realization of just how little additional work is forced on the receiver until the attack has become quite severe. While this is fully predicted by the theory, the degree and impact of it was not obvious until the tree could be visualized in realtime.

11.2 Sonic Demonstrator

While highly effective from many perspectives, the visualization demonstration is lacking in that it is not a tight analogy to the type of omnidirectional RF system envisioned as the eventual embodiment. To more closely model such a system in a way that still makes for a highly effective and portable demonstration, a sonic demonstration was also developed. The initial programs were developed jointly by Dr. Martin Carlisle[2] and Capt. Sean Butler[3] using digital signal processing algorithms provided by the author. This demo consists of two programs, a sender and a receiver. The sender takes a message and converts it to codewords one character at a time (each character is a separate message in terms of the codec) and produces a Windows

[1]Director, Academy Center for Cyberspace Research (ACCR), United States Air Force Academy, Colorado Springs, CO.

[2]Professor of Computer Science, United States Air Force Academy, Colorado Springs, CO.

[3]Instructor of Computer Science, United States Air Force Academy, Colorado Springs, CO.

Wave (WAV) file for that character and sends the file to the sound card. The result is a series of clicks emanating from the speaker. The second program records the sound present at the computer's microphone until told to stop. It then processes the recorded waveform to produce a packet, decodes the packet, and displays the recovered messages (characters) on the screen.

The "clicks" produced by the sending program consist of a 2ms fragment of a 2.8kHz sinusoidal tone (which was later reduced to 1ms). The choice of tone frequency was arbitrary with one goal being to use a low frequency so that low sampling rates could be used while being high enough to get at least a couple of complete cycles per mark. The processing chain and algorithms, which were essentially common to both the sonic and the early RF demos, is described in detail in Section 11.4.

The resulting demo allows quite effective realtime demonstrations by using two computers to send messages simultaneously while a third listens to the clearly overlapping series of apparently random, but identical, clicks. In order to permit the receiving computer to distinguish which computer a given message came from, the most significant bit (msb) in each ASCII character is set LO for one sender and HI for the other (mimicking a digital signature).

In addition to demonstrating the basic jam resistance properties, this demo also demonstrates that synchronization is not a problem (none of the three computers were synchronized in any fashion) and that oscillator differences were not overly critical (at least to the degree that the oscillators on different computers of different ages from different manufactures varied).

While this demo has been highly effective, it is not without significant limitations. It works well in moderately quite environments (it does not require highly silent surroundings) but is easily overcome by background noise. Part of this is intrinsic to the use of the built-in microphones and sound cards, which have a very limited dynamic range, and also to the automatic gain control built in to the sound card drivers.

None-the-less, more robust performance should be possible by using longer message lengths and also by incorporating interior checksum bits to combat the intrinsically higher noise floors.

A second sonic demo was developed to more thoroughly demonstrate more realistic configurations. In this demo, messages were typically 128 bits long and generally encoded into 6400-bit codewords that were transmitted at 1000 baud, thus requiring 6.4 seconds to transmit an entire packet. This demonstrator proved particularly valuable on two occasions. On the first, a new computer was added to the set and none of the messages it transmitted could be recovered by the receiving computer. A manual examination of the captured waveform data showed that the two oscillators were mismatched by 0.65% resulting in the terminal symbols being misaligned by 41 symbol periods. It was in response to this that the means of compensating for significant oscillator mismatch was developed and implemented. On the second occasion, it was observed during one demonstration that multiple copies of the transmitted messages kept appearing in the decoded output despite the fact that measures had been taken to suppress duplicate messages from contiguous packets. Upon modifying the decoder to add time stamps to each extracted message, it was discovered that the duplicate messages were consistently delayed by 11ms, which closely matched the expected flight time of an acoustic signal bouncing off a nearby sound-absorbing divider. Thus there was a multipath signal arriving such that it overlapped the direct signal by 99.8%. Though completely inadvertent, this clearly demonstrated the intrinsic high immunity of concurrent channels to multipath fading.

11.3 RF Demonstrators

While the visualization and sonic demos serve useful purposes, the entire intent in developing CCSS was to add capability to RF communications. Thus, the workhorse

demonstrators are software-defined radios built around the Ettus Research USRP (Universal Software Radio Peripheral) radio hardware. As of this writing, these demonstrators are pulse-based and use a static and predefined mark pattern. Most of the work has used a mark pattern that is simply a static high value in the baseband signal, which equates to simple On-Off Keying (OOK) of an otherwise unmodulated carrier. Other mark patterns, including LFM chirps, have also been used.

The most commonly used detection method to date has been a software-defined radiometer. The basic processing chain is shown in Figure

The receiver normally samples the down-converted data stream at four times the symbol rate. A common approach has been to use a matched filter at this higher sampling rate and follow it by a radiometer at the actual symbol rate. The time constant of the radiometer is generally set to 1.5 symbol periods. When a running statistic threshold is not used, the radiometer output is passed through a Schmitt-trigger discriminator. Several variants have been explored, all successfully, though relative performance at the margin has not been evaluated.

In addition to tests conducted at the Air Force Academy, these radios were used to collect data at U.S. Naval Air Weapons Station, China Lake during counter-UAV (Unmanned Aerial Vehicle) demonstration exercises in 2009 [99] and 2010 [100]. In these exercises, one radio was configured as a waveform recorder and mounted in an Osprey UAV operated by a group from the Air Force Research Lab (AFRL), Eglin Air Force Base, Florida. Another radio served as the friendly ground station transmitting a BBC-encoded waveform using a 10W amplifier and configured so that the mean transmitted power over the course of a packet was 10mW. The third radio was configured as the jammer and cycled through sets of packets having different numbers of valid messages overlaid that produced packet densities ranging from 10% to 90%. This radio was supplied to a team from the Tactical Electronic Warfare

Division of the Naval Research Lab, Washington D.C., who fed the signal into their 150W amplifier.

Due to a variety of hardware and logistical difficulties that affected all three teams, the quantity and quality of the captured data was very limited. None-the-less, it was demonstrated that low packet loss rates could be achieved, over ranges measured in thousands of meters, in the face of jamming signals having more than three orders of magnitude greater mean power at the receiver. In fact, by correlating the packet arrival times with the aircraft ground truth data, the primary cause of or lost packets appeared to be the result of aircraft maneuvers that pointed the underbelly mounted whip antenna toward the friendly station, which is the direction of highest loss.

11.4 Receiver DSP Algorithms

The signal processing that was developed for the sonic demonstrator was based on what was planned for the original RF demonstrator (which, at that time, was still some months in the future awaiting acquisition of suitable hardware). The original sonic demo sampled the PC's microphone at 8kSa/s while the pulses were nominally 2ms in duration, thus ideally sixteen samples were available for each pulse. In the early RF systems, the pulses were 1μs in duration and the input was I-Q sampled at 4MSa/s, yielding four samples per pulse.

The nominal processing chain is shown in Figure 11.4. Although RF terminology is used, the diagram is quite applicable to the sonic demo, in which the role of the Low Noise Amplifier (LNA) was served by a PC's microphone. The wideband filter was automatically provided by the PC's sound card based on the sampling rate. The same was true in the RF demo.

Therefore, the first digital signal processing (DSP) block that was custom-written was the radiometer, which first converted each sample in the signal stream into value

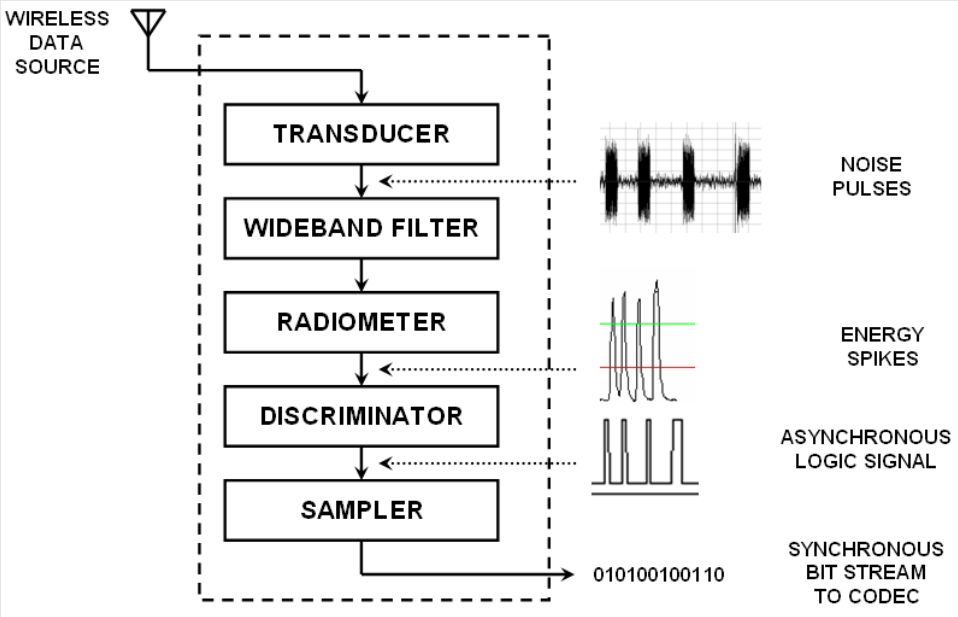

Figure 11.1: Receiver processing chain used in sonic and RF demos

proportional to the instantaneous energy. In the sonic demo, each input sample was simply squared while in the RF system each pair of I-Q samples at the input were squared and the summed. The energy signal was then passed through a first-order infinite impulse response (IIR) filter of the form

$$y[n] = \alpha x[n] + (1 - \alpha)y[n - 1] \tag{11.1}$$

where α was generally set to 0.25 for both systems, even though the sonic was working with the 2.8kHz carrier directly and the RF system was working with a baseband signal where the carrier had been downconverted to DC. It should be noted that parameters were intentionally not optimized since part of the goal was to demonstrate, if possible, the inherent ability to cope with sloppy tolerances.

The radiometer what followed with a Schmitt-triggered discriminator. The objective was to incorporate enough hysteresis so that once the radiometer output rose above the upper threshold, that the output would remain high through the variations while the mark signal was being received and not return to the low state until af-

ter the mark was complete and the radiometer had firmly decayed back to the level corresponding to a space. While this was quite beneficial in terms of the quality of the signal at the output of the discriminator, it didn't possess any specific ability to address the symbol misalignment issue and seemed prone to making mark errors under certain conditions; though, in practice, it worked quite well.

Since the signal stream was being sampled at greater than the symbol rate, it needed to be decimated to match the symbol rate before being passed to the decoder. The first decimator looked at the N samples out of the discriminator that spanned the symbol period and declared a mark if any of them were HI. While this appeared to work well, it was believed that it was vulnerable to mark errors due to both symbol misalignment and signal jitter.

The eventual solution (prior to the development of the running statistic threshold) was to decimate the signal before discriminating it using a peak detector running at the symbol rate. The peak detector simply produced the maximum of the most recent symbol window, which consisted of S samples with S generally being set to about $1.5N$. Thus, for the RF system which processed four samples per symbol, the value of each symbol was based on the largest of the most recent six samples. In a traditional system, this approach would be expected to produce unacceptable levels of intersymbol interference, but with a concurrent system is serves to greatly reduce the changes of a mark error while slightly increasing space error (i.e., false alarm) rate.

With the development of the peak-detecting decimator, there wasn't much purpose to the discriminator's hysteresis and therefore it was typically set to zero. At this point, all of these were combined into a single block that used two accumulators (since the peak-tracking period was typically longer than the symbol period. At the symbol rate, an accumulator and its associated counter were initialized to zero. Then, at the sample rate, the sample was converted to an equivalent energy (as previously

described) and compared to the contents of each active accumulator, replacing the value if it was greater. When an accumulator's counter reached the number of samples in the symbol window, it was discriminated by a simple threshold to determine the mark/space state of the bitstream sent to the decoder.

The steps have been modified only slightly to incorporate the running statistic threshold. Instead of discriminating the accumulator and putting a static mark/space into the decoder's data stream, the value of the accumulator is placed into the stream so that it can be discriminated against each threshold that applies to each individual packet that the symbol is a member of. It should be kept in mind that while a given symbol value will belong to C different packet frames, with C being on the order of one million, it will only be discriminated as needed and most packets will only require a small number of checks. Hence, the total number of thresholding operations is only a few times greater than the number associated with discriminating each sample once.

Chapter 12

Conclusion

12.1 Project Recap

This research indicates that concurrent code spread spectrum, and the theory of concurrent codes on which it is based, offers the ability to address potentially serious limitations in the field of jam resistant communications by eliminating the inherent key distribution difficulties associated with symmetric keys. The management and distribution of such keys, associated with traditional forms of spread spectrum as a consequence of the need for shared-secret spreading codes, is eliminated since CCSS dispenses with such secrets.

CCSS was not intended to perform better than traditional systems; in fact, it was expected that at least a modest amount of performance would have to be sacrificed as the price for addressing the key management problem. This dissertation demonstrates, at least in the specific case of barrage jamming and in the absence of error-correcting codes, that pulsed-BBC CCSS is expected to achieve a performance level somewhere between that of traditional frequency hop spread spectrum and direct sequence spread spectrum. This, in turn, indicates that the performance penalty associated with using concurrent code spread spectrum should be quite tolerable in

most applications. However, it must also be pointed out that BBC-based CCSS also comes with significant memory and processing burdens as well as latency issues associated with being intrinsically packet based. Fortunately, the processing requirements of BBC lend themselves to the application of high degrees of parallelism which can greatly reduce the latency involved. As always, these issues must all be considered when determining if CCSS is a suitable solution for a given application.

Fortunately, the processing and latency issues can be largely eliminated by using BBC in a hybrid mode in which a very limited number of BBC messages are used to effect a key exchange and then the exchanged session key is used to establish the shared-secret spreading code needed to continue the session using a traditional spread spectrum system.

Furthermore, CCSS offers the possibility of providing public-access systems a level of jam resistance comparable to that currently employed by shared-secret systems. However, much work remains to establish the viability of CCSS systems in actual practice. Perhaps most important, the basic analysis of the jam resistance of pulse-based CCSS needs to be extended to other forms of CCSS and to other forms of jamming. In addition, hardware implementations of the various CCSS options need to be designed, tested, and demonstrated.

12.2 Summary of Accomplishments

The BBC research project has contributed a number of accomplishments to the general body of scientific knowledge. From the standpoint of pure science, the most noteworthy is almost certainly a new coding theory, that of concurrent codes, that brings forth something that has not previously existed, namely a new family of superimposed codes that can be efficiently decoded. Given the disparate fields in which superimposed codes have found applications, despite not being efficiently decodable,

this potentially opens the door to many previously ill-suited applications. Supporting this new coding theory is considerable work on the behavior and characteristics of concurrent codes in general as well as the only existing concurrent code, namely BBC, including a security analysis.

From an applied science viewpoint, this work has again made possible something that has not previously existed, namely a means of attaining a jam-resistance omnidirectional radio link without relying on shared secrets. Again, much supporting work has been done both in analyzing the expected performance of concurrent code spread spectrum including a comparison to the performance of traditional spread spectrum systems in the presence of AWGN barrage jamming. In addition, considerable progress has been made in developing computationally efficient ways to implement a CCSS system in hardware, including efficient Golay correlators, running statistic threshold generators, task-specific hash functions, and highly parallelized decoders. Furthermore, numerous practical issues, as well as attack strategies, have been explored and each has been eliminated or mitigated through a variety of techniques ranging from algorithms for dealing with symbol misalignment and oscillator mismatch to decoder randomization and prioritization.

Beyond the theoretical development of both concurrent codes and concurrent code spread spectrum, several systems were developed to demonstrate these concepts ranging from simple visualization tools to sonic demonstrators that clearly showed the ability of the technology to work in high noise environments with simple and robust processing techniques. But, above all, radio frequency implementations were built that demonstrated all of the principal capabilities in real world environments, including airborne units flown on a naval weapons range and engaged by professional jamming systems.

12.3 Summary of Publications

To date, the BBC Research Project has published eight papers in the peer-reviewed literature and eight internal Air Force Academy technical reports. Peer-reviewed papers include two on concurrent coding theory and the BBC algorithm in general [25, 26], one on compensating for oscillator mismatch [76], one on the Inchworm concurrent hash function [79]. Other peer-reviewed papers focussed on ancillary topics related to BBC, including two on visualizing concurrent codes or applying concurrent codes to visual cryptography [101, 102], and one on using concurrent codes in computer science education [103], and one on the lessons learned as a result of presenting this highly non-traditional and interdisciplinary topic to audiences from difference disciplines and communities within those disciplines [104].

The Air Force Academy technical reports include the original paper on concurrent codes [24] and several subsequent tech reports on specific aspects of BBC including a security analysis of BBC [81], interstitial checksum bits [78], hardware implementation strategies [105, 82], the use of an efficient Golay correlator for mark detection [96], the running statistic threshold [80], and the UAV flight demonstrations at China Lake [99].

12.4 Independent Work Performed by Others

Three students at Auburn University have conducted research under the advisement of Dr. John Hamilton for which concurrent codes was central. Stephen Hamilton, while an M.Sc. student, implemented a PKI-based key exchange protocol using CAC-cards (Common Access Credential - also known as a "smart card") over a BBC-encoded channel [106]. For his Ph.D. research, Derek Sanders developed a BBC-based medium access control (BBC-MAC) protocol for mobile ad-hoc networks (MANET) [107] that dynamically adapts the codec parameters in response to noise conditions.

His work demonstrated that such BBC-based protocols can be implemented and are effective at combating channel noise.

Mark Kuhr, in his Ph.D. dissertation [108], also explored the use of BBC in adaptive cross-layer MANET protocols. As part of his work, Dr. Kuhr measured the relative level of jam resistance of a standard 802.11g protocol stack to a BBC-based stack with telling results. In his measurements, the BBC-based system lost only 4% of packets at jamming levels several times what was needed to prevent any successful packet transfers in the 802.11g system. In another set of measurements comparing the performance of a system using a typical Optimized Link State Routing (OLSR) protocol to one that incorporated a BBC-enabled physical layer, he demonstrated that the BBC-enabled system maintained forwarding queues containing about a dozen packets under conditions that drove the standard system to have several hundred packets and become so far behind that usable routing tables couldn't be maintained.

12.5 Potential Directions for Future Work

The spectrum of potential research topics related to concurrent codes is extremely rich and diverse. From an RF standpoint, numerous possible mark waveforms in addition to simple pulses have been identified, such as LFM chirps, frequency-time bins, and Golay sequences. The nominal performance of each of these needs to be analyzed. Similarly, detector options such as radiometers and Golay correlators need analytical attention. Likewise, a variety of practical issues, such as symbol misalignment and oscillator mismatch, need to have their effects quantified despite having been shown to be manageable. Then, for each combination of these that shows promise, optimal jamming waveforms need to be explored and analyzed. In particular, the feasibility of mark-erasure attacks requires attention for each of the mark/detector options.

Also of particular interest is the resistance to mark erasure in fading and multipath environments, especially those that result in echoes with sub-mark delays.

As noted in Chapter 10, the comparison to traditional spread spectrum techniques in this dissertation is not apples-to-apples, with some aspects biasing the analysis in favor of the traditional systems and others biasing it in favor of the concurrent code system. What is needed is to analyze communication systems based on each through a level of the stack at which they become black-box comparable as unreliable erasure channels.

In addition to the analyses described above, actual implementations of promising mark/detector combinations need to be realized, characterized, and compared to theoretical expectations. Included in this are field programmable gate array (FPGA) and application specific integrated circuit (ASIC) implementations of various parts of a concurrent codec. Of particular interest might be the signal quality requirements, such as linearity, of high power output amplifiers suitable for pulse-based BBC since, on the one-hand, the intended receiver may not require much linearity but regulatory requirements regarding spurious transmissions may be limiting factors.

On a broader front, CCSS has several potential applications that have barely been discussed, let alone explored in depth. Among these are radio-frequency identification (RFID), in which self-jamming is a known limiting issue. Another is the development of medium access control (MAC) protocols in which collisions are neither detected nor avoided, but simply tolerated. This work might build on the work already performed at Auburn University, or it might explore altogether different approaches.

From a basic science perspective, there are still many open questions, including whether other concurrent codes, besides BBC, can be created. One promising avenue involves identifying suitable Boolean monotone functions.

Clearly, the research landscape in concurrent codes is quite fertile and inviting both to researchers with a pure science interest as well as those with a strong inclination toward application development.

Bibliography

[1] J. Foster and L. Welch, "The evolving battlefield," Physics Today, vol. 53, no. 12, p. 31, Dec. 2000. [Online]. Available: "http://www.physicstoday.org/vol-53/iss-12/p31.html"

[2] J. Gowens and J. K. Young, "FY2001 annual report of the communications and networks consortium," Army Research Laboratory Collaborative Technology Alliance Program.

[3] "1996 Federal Radionavigation Plan," U.S. Gov't Printing Office, Jul. 1997.

[4] "2010 Federal Radionavigation Plan," U.S. Gov't Printing Office, Apr. 2011.

[5] Http://www.nsa.gov/ia/industry/gig.cfm?MenuID=10.3.2.2.

[6] "DEFENSE ACQUISITIONS - The Global Information Grid and Challenges Facing Its Implementation," U.S. Government Accountability Office, 2004.

[7] B. A. Forouzan, Cryptography and Network Security, 1st ed. McGraw Hill, 2008.

[8] B. Schneier, Applied Cryptography, 2nd ed. John Wiley & Sons, 1996.

[9] J. B. Anderson and S. Mohan, Source and Channel Coding: An Algorithmic Approach, 1st ed. Kluwer, 1991.

[10] R. Hill, A First Course in Coding Theory, 1st ed. Oxford Univ. Press, 1986.

[11] "Piccolo 433MHz Low Cost Data Radio," Product Sheet - RFInnovations Pty Ltd.

[12] A. El-Rabbany, Introduction to GPS: The Global Positioning System, 2nd ed. Artech House, 2006.

[13] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Trans. Inform. Theory, vol. IT-22, no. 6, pp. 644–654, 1976.

[14] M. E. Hellman, "An overview of public key cryptography," IEEE Communications Magazine, 50th Anniversary Commemorative Issue, pp. 42–49, May 2002.

[15] R. Merkle, "Secrecy, authentication, and public key systems," Ph.D. dissertation, Stanford University, 1979. [Online]. Available: "http://www.merkle.com/papers/Thesis1979.pdf"

[16] W. Diffie, "The first ten years of public-key cryptography," Proceedings of the IEEE, vol. 76, no. 5, pp. 560–577, May 1988.

[17] B. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," IEEE Communications, vol. 32, no. 9, pp. 33–38, Sep. 1994.

[18] J. Kohl, B. Neuman, and T. Ts'o, The Evolution of the Kerberos Authentication System. IEEE Computer Society Press, 1994.

[19] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in Advances in Cryptology - Pre-Proceedings of Eurocrypt '94, 1995.

[20] ——, "A secure and scalable group key exchange system," Information Processing Letters,, vol. 94, no. 3, pp. 137–143, 2005.

[21] Y. Desmedt and M. Burmester, "Towards practical proven secure authenticated key distribution," in Proceedings 1st ACM Conference on Computer and Communication Security, 1993, pp. 228–231.

[22] D. R. Stinson, T. van Trung, and R. Wei, "Secure frameproof codes, key distribution patterns, group testing algorithms and related structures," Journal of Statistical Planning and Inference, vol. 86, pp. 595–617, 2000.

[23] D. J. Torrieri, Principles of Secure Communication Systems, 2nd ed. Artech House, 1992.

[24] L. C. Baird, III, W. L. Bahn, and M. D. Collins, "Jam-resistant communication without shared secrets through the use of concurrent codes," United States Air Force Academy, Academy Center for Information Security, Tech. Rep. USAFA-TR-2007-01, 2007.

[25] L. C. Baird, III, W. L. Bahn, M. D. Collins, M. C. Carlisle, and S. C. Butler, "Keyless jam resistance," in Proc. 8th Annual IEEE SMC Information Assurance Workshop (IAW), Jun. 2007, pp. 143–150.

[26] W. L. Bahn, L. C. Baird, III, and M. D. Collins, "Jam-resistant communications without shared secrets," in Proc. 3rd International Conference on Information Warfare and Security (ICIW08), Apr. 2008, p. CD.

[27] A. Belouchrani and M. Amin, "Jammer mitigation in spread spectrum communications using blind source separation," Signal Processing, vol. 80, no. 4, pp. 723–729, Apr. 2000.

[28] L. B. Milstein, "Interference rejection techniques in spread spectrum communications," Proc. IEEE, vol. 76, no. 6, pp. 657–671, Jun. 1998.

[29] G. J. Saulnier, Z. Ye, and M. J. Medley, "Performance of a spread-spectrum OFDM system in a dispersive fading channel with interference," in Proc. MILCOM Conf., 1998, pp. 679–683.

[30] J. P. F. Glas, "On multiple access interference in a DS/FFH spread spectrum communication system," in Proc. of the Third IEEE International Symposiumon Spread Spectrum Techniques and Applications, Oulu, Finland, Jul. 1994.

[31] E. G. Kanterakis, "A novel technique for narrowband/broadband interference excision in DS-SS communications," in MILCOM '94, vol. 2, 1994, pp. 628–632.

[32] L. Li and L. Milstein, "Rejection of pulsed CW interference in PN spread-spectrum systems using complex adaptive filters," IEEE Trans. Commun., vol. COM-31, pp. 10–20, Jan. 1983.

[33] L. B. Milstein, "Interference suppression to aid acquisition in direct-sequence spread-spectrum communications," IEEE Transactions on Communications, vol. 36, no. 11, pp. 1200–1207, Nov. 1988.

[34] H. V. Poor and L. A. Rusch, "Narrowband interference suppression in spread spectrum CDMA," IEEE Personal Communication Magazine, vol. 1, pp. 14–27, Aug. 1994.

[35] M. Davis and L. Milstein, "Implementation of a CDMA receiver with multiple-access noise rejection," in Proceedings of the Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '92), Oct. 1992, pp. 103–107.

[36] C. Bergstrom and J. Chuprun, "Optimal hybrid frequency hop communication system using nonlinear adaptive jammer countermeasures and active fading mit-

igation," in IEEE Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration, vol. 6, Nov. 1998, pp. 3426–3431.

[37] I. Bergel, E. Fishler, and H. Messer, "Low complexity narrow-band interference suppression in impulse radio," in Proceedings of the 2003 International Workshop on Ultra Wideband Systems (IWUWBS), Oulu, Finland, Jun. 2003.

[38] ——, "Narrow-band interference suppression in time-hopping impulse-radio systems," in Proceedings of the Conference on Ultra Wideband Systems and Technologies, Baltimore, MD, May 20-23, May 2002, pp. 303–307. [Online]. Available: http://citeseer.ist.psu.edu/bergel02narrowband.html

[39] R. Blazquez and A. P. Chandrakasan, "Architectures for energy-aware impulse UWB communications," ICASSP, Mar. 2005.

[40] W. Kautz and R. Singleton, "Nonrandom binary superimposed codes," IEEE Transactions on Information Theory, pp. 363–377, 1964.

[41] D. Danev, "Some constructions of superimposed codes in euclidean spaces," Discrete Applied Mathematics, vol. 128, no. 1, pp. 85–101, May 2003.

[42] G. Ahlstrm and D. Danev, "A class of superimposed codes for CDMA over fiber optic channels," Linkoping University, Tech. Rep. LiTH-ISY-R-2543, 2003.

[43] P.-O. Anderson, "Superimposed codes and Bn-sequences," Linkoping University, Tech. Rep. LiTH-ISY-I-1108, 1990.

[44] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[45] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," in SIGCOMM '05: Proceedings of the 2005 conference on Applications,

technologies, architectures, and protocols for computer communications. New York, NY, USA: ACM Press, 2005, pp. 181–192.

[46] A. Clementi, A. Monti, and R. Silvestri, "Distributed broadcast in radio networks of unknown topology," Theor. Comput. Sci., vol. 302, no. 1-3, pp. 337–364, 2003. [Online]. Available: "http://dx.doi.org/10.1016/S0304-3975(02)00851-4"

[47] P. Erdos, P. Frankl, Z. Furedi, and Z. Furedi, "Families of finite sets in which in which no set is covered by the union of r others," Israel Journal of Mathematics, vol. 51, pp. 75–89, 1985.

[48] M. Ruszinko, "On the upper bound of the size of the R-cover-free families," in Proceeding of the 1993 IEEE International Symposium on Information Theory, 1993, p. 367.

[49] A. D'yachkov, V. Lebedev, P. Vilenkin, and S. Yekhanin, "Cover-free families and superimposed codes: Constructions, bounds, and applications to cryptography and group testing," in IEEE International Symposium on Information Theory, 2001.

[50] D. R. Stinson and R. Wei, "Generalized cover-free families," Discrete Mathematics, vol. 279, pp. 463–477, 2004.

[51] R. Wei, "On cover-free families," Lakehead University, Tech. Rep., 2006.

[52] A. D'yachkov, A. Macula, D. Torney, P. Vilenkin, and S. Yekhanin, "New results in the theory of superimposed codes," in Proceedings of International Conf. on Algebraic and Combinatorial Coding Theory (ACCT), 2000, pp. 126–136.

[53] S. Yekhanin, "Sufficient conditions of existence of fix-free codes," IEEE International Symposium on Information Theory, Jun. 2001.

[54] A. Dyachkov and V. Rykov, "Optimal superimposed codes and designs for renyi's search model," Journal of Statistical Planning and Inference, vol. 100, pp. 281–302, 2002.

[55] A. de Bonis and U. Vaccaro, "Constructions of generalized superimposed codes with applications to group testing and confict resolution in multiple access channels," Theoretical Computer Science, vol. 306, pp. 223–243, 2003.

[56] A. D'yachkov, P. Vilenkin, and S. Yekhanin, "Upper bound on the rate of superimposed (s,l) codes based on Engel's inequality," in Proceedings of the International Conf. on Algebraic and Combinatorial Coding Theory (ACCT), 2002, pp. 95–99. [Online]. Available: "http://theory.csail.mit.edu/ yekhanin/Papers/dvy-02.pdf"

[57] S. Yekhanin, "Some new constructions of optimal superimposed designs," in Proceedings of International Conf. on Algebraic and Combinatorial Coding Theory, 1998, pp. 232–235. [Online]. Available: "http://theory.csail.mit.edu/ yekhanin/Papers/acct6.pdf"

[58] A. de Bonis and U. Vaccaro, "Efficient constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels." in ESA, 2002, pp. 335–347.

[59] T. Erickson and L. Gyorfi, "Superimposed codes in rn," Linkoping University, Tech. Rep. LiTH-ISY-I-0818, 1986.

[60] T. Huang and C. wen Weng, "A note on decoding of superimposed codes," Journal of Combinatorial Optimization, vol. 7, pp. 381–384, 2003.

[61] D. Awduche and A. Ganz, "MAC protocol for wireless networks in tactical environments," in IEEE Military Communications Conference, MILCOM-96, Oct. 1996.

[62] L.-I. Alfredsson, "A class of superimposed codes for cdma over fiber optic channels," Linkoping University, Tech. Rep. LiTH-ISY-I-1045, 1989.

[63] Fidel Cacheda and Ángel Viña, "Superimposing codes representing hierarchical information in web directories," in WIDM, 2001, pp. 54–60.

[64] Y. Desmedt, "A high availability internetwork capable of accomodating compromised routers," BT Technology Journal, vol. 24, no. 4, pp. 77–83, Jul. 2006.

[65] J. Komlos and A. Greenberg, "An asymptotically nonadaptive algorithm for conflict resolution in mutliple-access channels," IEEE Transactions on Information Theory, vol. IT-31, no. 2, pp. 302–306, Mar. 1985.

[66] Http://www.nist.gov/dads/HTML/superimposedCode.html.

[67] D. Du and F. Hwang, Combinatorial Group Testing and Its Applications. World Scientific, 1993.

[68] R. Dorfman, "The detection of defective members of large populations," Annals of Mathematical Statistics, vol. 14, pp. 436–440, 1943.

[69] R. Casey and et. al., Eds., Punched Cards, Their Applications to Science and Industry. Reinhold Publishing Corp., 1958, ch. 10 - An application of random codes for literature searching.

[70] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking most frequent items dynamically," in Proceedings of ACM Principles of Database Systems, 2003, pp. 296–306. [Online]. Available: "http://acm.org/sigmod/pods/proc03/online/210-cormode.pdf"

[71] P. de Laval and S. Abdu-Jabbar, "Decoding of superimposed codes in multiaccess communication," in Conference Proceedings on Area Communication, EUROCON 88, Jun. 1988, pp. 154–157.

[72] S. Abdul-Jabbar and P. de Laval, "Constant weight codes for multiaccess channels without feedback," in Conference Proceedings on Area Communication, EUROCON 88, Jun. 1988, pp. 150–153.

[73] P. de Laval, "Decoding of superimposed codes," Linkoping University, Tech. Rep. LiTH-ISY-R-0958, 1988.

[74] ——, "Sliding window reduced search decoding for superimposed codes," Linkoping University, Tech. Rep., 1989.

[75] ——, "Superimposed code reservations systems- description and examples of possible implementation principles," Linkoping University, Tech. Rep., 1989.

[76] W. L. Bahn, L. C. Baird, III, and M. D. Collins, "Oscillator mismatch and jitter compensation in concurrent codecs," in Proc. 2008 IEEE Military Communications Conference (MILCOM08), Nov. 2008, p. CD.

[77] A. Leon-Garcia and I. Widjaja, Communication Networks: Fundamental Concepts and Key Architectures, 2nd ed. McGraw Hill, 2004.

[78] W. L. Bahn and L. C. Baird, III, "Extending critical mark densities in concurrent codecs through the use of interstitial checksum bits," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-02, Dec. 2008.

[79] L. C. Baird, III, M. C. Carlisle, and W. L. Bahn, "Unkeyed jam resistance 300 times faster: The inchworm hash," in Proc. 2010 IEEE Military Communications Conference (MILCOM10), Nov. 2010, p. CD.

[80] L. C. Baird, III and W. L. Bahn, "An O(log n) running median or running statistic method, for use with BBC jam resistance," United States Air Force

Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-03, Nov. 2009.

[81] ——, "Security analysis of BBC coding," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-01, Dec. 2008.

[82] ——, "Parallel BBC decoding with little interprocess communication," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2009-ACCR-01, Nov. 2009.

[83] R. L. Peterson, R. E. Ziemer, and D. E. Borth, Introduction to Spread Spectrum Communications. Prentice Hall, 1995.

[84] S. Gyori, "Signature coding over multiple access OR channel," in Winter School on Coding and Information Theory, 2003.

[85] C. C. Jonsson, "Anti-jamming on the or-channel," in 1994 IEEE International Symposium on Information Theory, Trondheim, Norway, Jul. 1994, pp. 480–480.

[86] R. Blazquez, P. Newaskar, and A. Chandrakasan, "Coarse acquisition for ultra wideband digital receivers," in Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. IV, Hong Kong, China, Apr. 2003, pp. 137–140.

[87] K. J. Negus, J. Waters, J. Tourrilhes, C. Romans, J. Lansford, and S. Hui, "HomeRF and SWAP: Wireless networking for the connected home," ACM Mobile Computing and Communications Review, vol. 2, no. 4, pp. 28–37, Oct. 1998.

[88] P. Newaskar, R. Blazquez, and A. Chandrakasan, "A/D precision requirements for an ultra-wideband radio receiver," in SIPS 2002, San Diego, CA, Oct. 2002, pp. 270–275.

[89] M. Z. Win and R. A. Scholtz, "Impulse radio: how it works," IEEE Communications Letters, vol. 2, no. 2, pp. 36–38, Feb. 1998.

[90] R. J.-M. Cramer, R. A. Scholtz, and M. Z. Win, "Evaluation of an ultra-wideband propagation channel," IEEE Transactions on Antennas and Propagation, vol. 50, no. 5, pp. 561–570, May 2002.

[91] M. Z. Win and R. A. Scholtz, "On the robustness of ultra-wide bandwidth signals in dense multipath environments," IEEE Communications Letters, vol. 2, no. 2, pp. 51–53, Feb. 1998.

[92] ——, "On the energy capture of ultrawide bandwidth signals in dense multipath environments," IEEE Communications Letters, vol. 2, no. 9, pp. 245–247, Sep. 1998.

[93] ——, "Ultra-wide bandwidth time-hopping spread-spectrum impulse radio for wireless multiple-access communications," IEEE Transactions on Communications, vol. 48, no. 4, pp. 679–691, Apr. 2000.

[94] S. Yilmaz and I. Tekin, "Ultra-wideband n-bit digitally tunable pulse generator," in Ultra-Wideband, 2005. ICU 2005. 2005 IEEE International Conference on, Sep. 2005, pp. 438–441.

[95] M. L. Skolnik, Introduction to Radar Systems, 2nd ed. McGraw Hill, 1980.

[96] L. C. Baird, III and W. L. Bahn, "An efficient correlator for implementations of BBC jam resistance," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-02, Nov. 2009.

[97] B. G. Lee and B.-H. Kim, Scrambling Techniques for CDMA Communications, 1st ed. Kluwar, 2001.

[98] R. E. Ziemer and R. L. Peterson, Introduction to Digital Communication, 2nd ed. Prentice Hall, 2001.

[99] W. L. Bahn, "A field demonstration of unkeyed jam resistance," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2010-ACCR-04, Dec. 2010.

[100] ——, "A field demonstration of unkeyed jam resistance - part II," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2011-ACCR-01, Oct. 2011.

[101] D. L. Schweitzer, L. C. Baird, III, and W. L. Bahn, "Visually understanding jam resistant communication," in Proc. 3rd Intl. Workshop on Visualization for Computer Security (VizSec), Oct. 2007.

[102] L. C. Baird, III, D. L. Schweitzer, W. L. Bahn, and S. Sambasivam, "A novel "Visual Cryptography" coding system for jam resistant communications," Journal of Issues in Informing Science and Information Technology, vol. 7, pp. 495–507, 2010.

[103] W. L. Bahn, L. C. Baird, III, and M. D. Collins, "The use of concurrent codes in computer and digital signal processing education," Journal of Computing Sciences in Colleges, vol. 23, no. 1, pp. 174–180, Oct. 2007.

[104] ——, "Impediments to systems thinking: Communities separated by a common language," in Proc. 4th Intl. Conf. on Cybernetics, Information Technologies, Systems and Applications (CITSA), vol. III, Jul. 2007, pp. 122–127.

[105] W. L. Bahn and L. C. Baird, III, "Hardware-centric implementation considerations for BBC-based concurrent codecs," United States Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-03, Dec. 2008.

[106] S. S. Hamilton and J. A. Hamilton, Jr., "Secure jam resistant key transfer: Using the DoD CAC card to secure a radio link by employing the BBC jam resistant algorithm," in Proc. 2008 IEEE Military Communications Conference (MILCOM08), Nov. 2008, p. CD.

[107] D. T. Sanders, "An adaptive single-hop medium access control layer for noisy channels," Ph.D. dissertation, Auburn University, 2009.

[108] M. G. Kuhr, "An adaptive jam-resistant cross-layer protocol for mobile ad-hoc networks in noisy environments," Ph.D. dissertation, Auburn University, 2009.

Appendix A

Glossary of Terms

attack packet The collection of marks inserted into a packet by an attacker.

attacker The party attempting to prevent the receiver from receiving the sender's message.

autohallucination A codeword that, by itself, covers other codewords.

BBC A concurrent coding algorithm based on sequential partial encoding/decoding.

checksum bits In BBC, zero bits that are appended to the end of a message or otherwise inserted into a message string prior to encoding. These serve much the same purposes as traditional checksum bits in other applications.

codebook The collection of all valid codewords.

codec In general, a device or algorithm that codes messages into codewords and decodes messages from packets.

codeword The binary encoding of a message that is to be transmitted.

concurrent codec A codec specifically designed to work with packets that contain concurrent codewords.

covered codeword A codeword all of whose marks are present in the packet being decoded.

critical density The packet density at which decoding is expected to produce an exponential number of terminal hallucinations.

density The fraction of marks in a codeword or packet.

hallucination A message produced by decoding a packet that was not intentionally inserted into the packet.

information bits In BBC, the bits of the padded message that can take on either value, as opposed to checksum bits which are always a known value of zero.

legitimate message A message that has been encoded and inserted into a packet by the genuine sender.

mark A binary symbol having (ideally) zero transmission error probability.

message A binary bit string that is to be conveyed from sender to receiver.

multiple access OR channel An environment were multiple codewords are combined via bitwise-OR.

packet A collection of concurrent codewords, i.e., codewords that have been combined in a multiple access OR channel.

receiver The intended recipient of a sender's message.

rogue message A message that has been encoded and inserted into a packet by a hostile player (attacker).

sender The party attempting to send a message to a receiver (a.k.a., legitimate sender).

space A binary symbol potentially having a significant transmission error probability.

terminal hallucination A message that has survived the decoding process up to the point of decoding the checksum bits.

valid codeword A binary bit string that can result from encoding a message.

valid message A message that has been encoded and inserted into a packet, either by the genuine sender or by a hostile player (attacker) (i.e., not an hallucination).

working hallucination A message that exists at some intermediate point in the decoding process.

Appendix B

Glossary of Symbols

Table B.1: Glossary of Symbols

Codec Parameters		
L	=	message length (bits)
F	=	expansion factor
C	=	codeword length (bits)
K	=	number of terminal checksum bits
S	=	number of interstitial checksum bits per fragment
D	=	number of information bits per fragment
B	=	number of padded message bits per encoding block
Y	=	number of marks generated per encoding block
Q	=	number of marks that must be found to declare a block found
Codeword and Packet measures		
d	=	degree of codeword/packet (number of marks)
μ_c	=	codeword mark density
μ_p	=	packet mark density
Performance-related Parameters		
M_S	=	Messages sent by the legitimate sender
M_A	=	Messages (or their equivalent) sent by the attacker
M_R	=	Messages output by the receiving codec
M_H	=	Messages that are hallucinations
ζ_A, ζ_R	=	Relative effort of attacker/receiver

Appendix C

Expanded Explanations of Terms

This appendix provides fuller explanations of many of the terms used in this work and relates them to each other. While some of this is duplicated within the body of the text, it is believed that presenting them here in collected form will prove beneficial by minimizing the need to seek out the relevant descriptions within the text.

C.1 BBC Codec Parameters - L, F, C, K, S, D, B, Y, Q

A BBC-based codec, like many codecs, takes L-bit messages and generates C-bit codeword. As with error-correcting codecs, C is larger than L but, unlike error-correcting codes that use the additional bits to incorporate redundant information that can be used to detect errors and reconstruct the original codeword from a corrupted packet, in concurrent coding the additional bits are used to create sparseness in the codeword so as to accommodate the addition of undesired marks. None-the-less, a fair degree of common purpose, at least conceptually, exists and, like error-correcting codes, the greater the ratio between the codeword length and the message length, the greater the ability of the code to achieve its goal of recovering the correct message from a

corrupted packet. This is such a key factor in the capability of the code that it is the dominant parameter in the performance of the code and is the *expansion factor*, F.

$$L \equiv \text{message length (bits)}$$

$$C \equiv \text{codeword length (bits)}$$

$$F = \text{expansion factor} \equiv \frac{\text{codeword length}}{\text{message length}} = \frac{C}{L}$$

The expansion factor is always based on the number of bits in the raw message and does not reflect any additional bits added to the message prior to encoding.

Another key codec parameter is how many marks appear in each codeword. This is defined as the *degree*, d, of the codeword (e.g., a codeword of degree 62 has 62 marks in it). Like the codeword length, this is best expressed in terms of the message length, hence the *mark factor*, s, is the ratio of the nominal number of marks in the codeword to the length of the original message.

$$s = \text{mark factor} \equiv \frac{\text{marks in codeword}}{\text{message length}} = \frac{d}{L}$$

In some contexts it is important to understand that a codeword may have a different *actual degree* than its *nominal degree*. This is particularly the case for stochastic generating algorithms such as BBC that may produce degenerate marks (i.e., multiple marks placed at the same location within the same codeword). Generally the context of the discussion is sufficient to determine which meaning is relevant.

The BBC algorithm incorporates the ability to reject hallucinations as part of the normal decoding process by appending checksum bits to the end of the message prior to encoding it. The checksum bits are simply zero bits and are not dependent on the contents of the message, however they place additional marks in the codeword at locations that are a function of the entire message, and hence serve the same

purpose as checksum bits in more traditional contexts. The number of checksum bits, k, that are appended to the message determines what expected fraction of the terminal hallucination will survive to become hallucinations in the final message list for a particular packet density.

K = number of zero bits appended to message to serve as checksum bits

An extension of the basic BBC algorithm that permits the decoder to continue working with very dense packets is to partition the original message into a series of fragments and insert additional checksum bits to the beginning of each fragment. The parameter S is the number of checksum bits that are prepended to each fragment prior to encoding. Like the checksum bits at the end of the message, the interstitial checksum bits are simply zero bits - a different name is used merely to distinguish the two. The parameter D is the number of message bits in each fragment (the last fragment may have fewer if the message length is not an integral multiple of the fragment length).

S = number of zero bits prepended to each message fragment

D = number of message bits in each (full) message fragment

C.2 Packet Parameters - P, M, μ

A packet, P, is a collection of codewords (and/or packets) that have been combined by performing a bitwise logical-OR'ing of all the included codewords. Because of this,

codewords and packets share many of the same descriptors, including message load, M, and mark density, μ.

The message load of a packet is the number of messages whose codewords have been combined to produce that packet. Depending on the context, it may include or exclude hallucinations.

The mark density is the fraction of bits in the codeword or packet that contain marks. In other words, it is the degree of the codeword or packet divided by the codeword or packet length.

$$\mu = \text{mark density} \equiv \frac{\text{marks in codeword}}{\text{codeword length}} = \frac{d}{C} = \frac{d}{FL} \tag{C.1}$$

In the case of individual codewords where the degree is dictated by the mark factor, this can also be expressed, at least as an expectation value, as

$$\mu_c = \frac{d}{C} = \frac{Ls}{LF} = \frac{s}{F} \tag{C.2}$$

C.3 Sender, Receiver, and Attacker - S, R, A

Most discussions will require three actors: a sender, a receiver, and an attacker. We could use the archetypical actors Alice and Bob from the cryptographic community, adding in the less-well-known Mallory as our malicious, active attacker, but we have chosen not to. As such, parameters will frequently be subscripted with an S, an R, or an A to indicate that they apply to the sender, receiver, or attacker respectively.

C.4 Attacker and Receiver Effort - E, ζ

The end goal of much of the analysis in this work is to compare the amount of effort that the attacker can force the receiver to perform as a function of the amount

of energy expended in the attack. As in most other treatments of communications jamming we will normally want to, at least eventually, normalize the receiver's effort to the amount of effort they would expend if there were no jamming to contend with.

Our definition of energy, E, is necessarily rather abstract so as to establish a well defined measure that can be applied consistently to the various concurrent codecs being analyzed, but the intent is that the relevant effects in practical applications should scale approximately in accordance.

The relative energy used by the attacker is defined as the ratio of the marks in the attack packet to the marks in the sent packet. In transmission schemes such as pulse-based ultra-wideband, this definition is particularly appropriate. In other schemes it may be less so.

$$\zeta_A \equiv \frac{E_A}{E_S} = \frac{d_A}{d_s} = \frac{\mu_A}{\mu_S} \tag{C.3}$$

The relative energy required by the receiver is defined as the ratio of the computational work needed to extract and process all of the messages (M_R) to the amount of computational work that would be required if no jamming were taking place($M_R^{'}$). In most cases, $M_R^{'}$ will be equal to the number of genuine messages sent (M_S). Our proxy for computational effort will be the number of messages that are contained in a packet, hence

$$\zeta_R \equiv \frac{M_R}{M_R^{'}} = \frac{M_R}{M_S} \tag{C.4}$$

Depending on the details of the codec, the relevant point at which to measure M_R may not be at the codec's output. In general, it will be at the point in the processing chain that dictates the capacity limits.

Appendix D

Running Statistic Threshold

The running statistic threshold is a data structure that has been developed to enable a BBC receiver to quickly update the threshold value that corresponds to a particular packet density. Details regarding the theory and implementation can be found in [80]. The discussion here is only intended to familiarize the reader with the general concept and mechanics involved.

The running statistic structure is built on a common data structure known as a "heap", which is also known as a "priority queue". First we will look as a typical heap and see how it is maintained. Then we will explore how two heaps can be combined to track a particular statistic, such as the 33rd percentile.

A heap is a tree structure in which the only requirement is that each node be at least as large as any of its children (or, equivalently, be no larger than its parent). This guarantees that the root node is largest value in the structure; being more precise, it guarantees that there are no other nodes in the tree that are larger than the root node, since there may be others that are just as large. Its important to understand that this about is all that can be said about a heap structure, other than that the smallest value (or one of them, if there are ties) is in one of the leaf nodes. In particular, as Figure D illustrates, it is entirely possible for one of the leaf nodes to be just as large

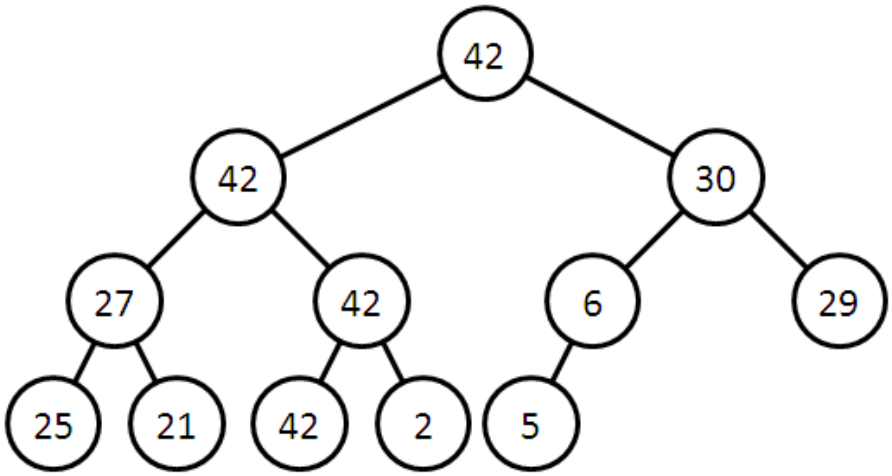

Figure D.1: Generic heap structure.

as the root node without violating the heap's constraints. In other words, a node's position in the tree tells us almost nothing about the relative order of that node in sorted list of the heap's contents.

Typically, because of the applications that heaps usually find themselves in, the root node is removed and replaced with a new value. That new value may or may not be as large as its children. If it is, then the heap is still valid, but if it isn't, then the heap must be "healed". This is done by comparing the new value of the node to the largest of its children and, if it is smaller, swapping them. By only looking at the larger of the children, we are guaranteed that the new value of the parent is at least as large as the other children and, hence, no further swaps involving it will be necessary. However, if a swap occurs, then the node were the new value was put may still be smaller than one of its new children. The process of comparing to the largest of the children as swapping, if necessary, is continued until a swap is not needed (or until the value has "sunk" to a leaf node). Once this happens, the tree is healed and represents a valid heap structure once again.

In a running statistic threshold, we are only interested in using the value of the root node, not in removing it. Instead, as the packet window moves along the data

stream, we are interested in removing the oldest node and replacing it with the newly added node; therefore the node that is updated could be located anywhere within the structure. The only modification this requires is that the new value may have to "rise" or it may have to sink. If it needs to rise, it will only need to rise to its proper location while if it needs to sink, it will only need to sink. In other words, once the direction of travel is determined, no (further) checks related to the other direction need be made. The simplest check is to see if the new node must be swapped with its parent. If it does, swap the nodes and check the new node against its new parent, swapping if necessary, until the heap is healed. If it doesn't have to be swapped with it initial parent, then we must check to see if it must be swapped with the largest of its initial children and, if so, proceed as described previously to heal the heap.

Up to this point, we have talked about using a heap in which the root node is the largest value stored in the heap. However, we can reverse this as have it represent the smallest value in the heap by simply requiring that each node be no smaller than its parent (or, equivalently, no larger than either of children). We will call the former heap structure a "maximal heap" and the reversed structure a "minimal heap".

No consider the structure shown in Figure D. Here we have combined a maximal heap on the bottom and a minimal heap on the top. They share a common root node, which is the node identified as the threshold. The processing is identical to what has been described previously with the additional being that if a node in the lower heap rises to the threshold, then we must switch over and see if it must continue to rise in the upper heap as well. Similarly, nodes that fall to the threshold node from the upper heap may need to continue falling into the lower heap.

To set the threshold at a particular statistic, all that is necessary is to partition the nodes correctly between the two heaps. In the case of standard BBC, the desired threshold is the one that results in the packet density being as close to 1/3 without exceeding it. Since there is one node for each bit in the packet, the lower heap

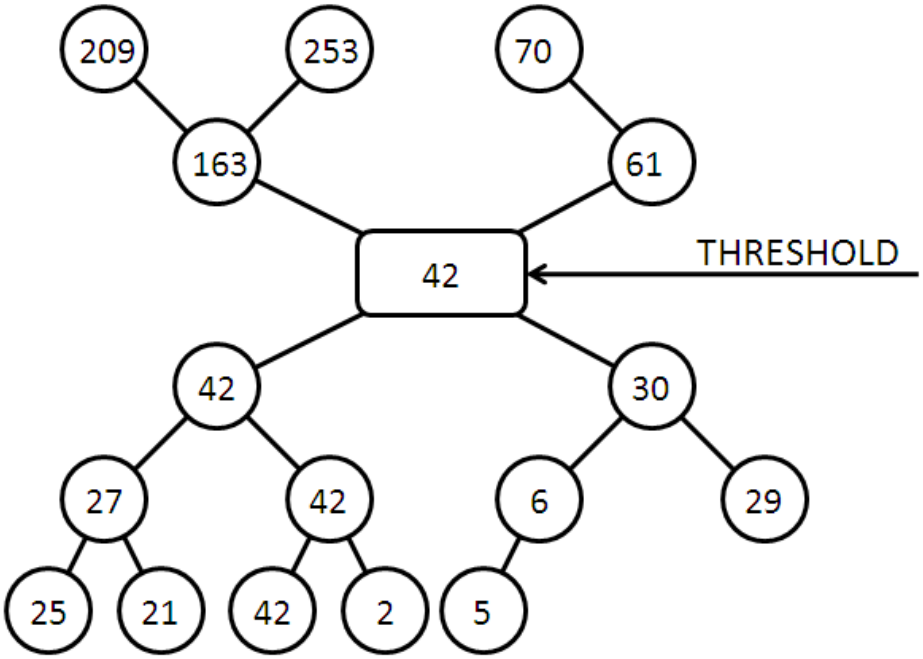

Figure D.2: Dual heap structure used in the running statistic threshold.

(including the threshold node) should contain

$$N_L = \lceil \frac{2}{3} C \rceil \tag{D.1}$$

nodes with the remainder going in the upper heap.

The criterion for determining if a particular bit in the packet is a mark is whether its value exceeds the threshold; it is important that values that are equal to the threshold be considered spaces. To see why this is true, consider a structure in which every node has the same value. If equality with the threshold was the criterion, then the resulting packet would have a density of 100%, which would overwhelm the decoder. Instead, in this case, we want the packet to have a density of 0%. By using the strictly-greater-than criterion, the threshold is truly a percentile in that only those packet bits that are at or above the 2/3 percentile (using the partition described here) will be counted as marks.

As might already be apparent, we need to keep track of not only the values of the nodes, but also what position in the packet they correspond to since we have to remove the oldest and replace it with the newest. The cleanest way to do this is with three arrays. One array stores the time-ordered values of the received bit stream. In most implementations, this will likely be a ring register holding a few packets worth of data to maximize resource availability to the various parts that need to read and write to the packet buffer. A second, parallel array would stored indices into the heap structure to the node corresponding to the value in the first array. The heap structure itself would not store the actual node values, but rather an index back to the two parallel array. Taken together, this results in a statically allocated doubly-link list.

With regards to the implementation of the heaps themselves, the recommended approach is to place the heaps in two adjacent arrays with one heap going forward in memory from the root and the other going backward (i.e., using negative indices). The nodes of the heap are stored one level at a time. The addressing then becomes very straight-forward (see [80] for details). For example, given the node in position n, the two children are at locations $2n+1$ and $2n+2$. Similarly, the parent is located at $(n-1)/2$ (provided integer division is used).

Appendix E

Q-Function

The Q-function is widely used in the communications and other communities, but is probably unfamiliar to many readers. The definition of the Q function is

$$\mathrm{Q}(x) \triangleq \begin{cases} \int_x^{+\infty} Z(u)du; & x \geq 0 \\ 1 - \mathrm{Q}(-x); & x < 0 \end{cases} \tag{E.1}$$

where $\mathrm{Z}(x)$ is the normalized Gaussian probability density function (pdf) defined by

$$\mathrm{Z}(x) \triangleq \frac{e^{-x^2/2}}{\sqrt{2\pi}} \tag{E.2}$$

Most readers are probably familiar with the pdf for a Gaussian having mean μ and variance σ^2, namely

$$f(v) = \frac{e^{-(v-\mu)^2/(2\sigma^2)}}{\sqrt{2\pi\sigma^2}} \tag{E.3}$$

Thus, it is clear that the Z-function is merely the pdf of a zero mean Gaussian having unity variance. With this in mind, the value returned by $Q(x)$ is the probability that a Gaussian random variable will take on a value greater than x standard deviations from the mean. For clarity, note that if $x < 0$, then the probability will be greater

than 1/2 because x will lie below the mean and $\mathrm{Q}(x)$ is the probability that the random value is greater than x.

For readers that need to evaluate the Q-function but do not have access to a mathematical tool that supports this function, several approximations exist. Two in particular may be found in [83]. For those that have access to tools supporting the standard error functions, such as most spreadsheet programs, another alternative is to note that the Q-function is merely a rescaling of the complementary error function, defined as

$$\mathrm{erfc}(x) \triangleq 1 - \mathrm{erf}(x) \tag{E.4}$$

where

$$\mathrm{erf}(x) \triangleq \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \qquad x \geq 0 \tag{E.5}$$

Straightforward manipulations yield

$$\mathrm{Q}(x) = \begin{cases} 1 - \frac{1}{2}\mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right) & x < 0 \\ \frac{1}{2}\mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right) & x \geq 0 \end{cases} \tag{E.6}$$

and

$$\mathrm{Q}^{-1}(y) = \begin{cases} \sqrt{2}\mathrm{erfc}^{-1}(2y) & 0 \leq y \leq 0.5 \\ -\sqrt{2}\mathrm{erfc}^{-1}(2(1-y)) & 0.5 < y \leq 1.0 \end{cases} \tag{E.7}$$

Appendix F

Superimposed Codec Characteristic

The performance of a generic superimposed concurrent codec can be estimated using the following assumptions:

1. Messages are L-bits long.

2. Codewords are F times the length of a message.

3. There are η message bits per codeword mark.

4. The "cost" incurred by a receiver is proportional to the number of complete messages decoded.

5. The "cost" incurred by a transmitter is proportional to the number of marks transmitted.

6. Attack packets are generated by superimposing complete messages.

7. Codewords are independent of each other.

All but the first two assumptions are problematic to varying degrees. Depending on how codewords are assigned, some messages may have more marks than others, thus η may not be a fixed value. We will assume that it either is or is at least a good approximation. The cost assumptions are quite arbitrary, but anything more realistic requires knowledge of the details of a particular implementation or operating environment; however, many of the real costs will be more-or-less proportional to these costs, and therefore they serve as a useful abstract metric. The assumption that attack packets are produced by superimposing complete messages probably does not represent the optimal attack strategy, but it is the only reasonable assumption that can be made for a generic superimposed codec; given a particular encoding algorithm, it is almost certainly the case that somewhat better attack packets can be crafted by assembling them on a mark-by-mark basis, but any practical encoding scheme will have to limit the relative effectiveness of such optimal packets to being only modestly better than random message attack packets. Finally, assuming that all of the codewords are independent of each other is almost certainly not reasonable; if it were, then the only way to decode the packet would be via a brute force exhaustive search of the message space. However, the analysis for such a "random codebook" still provides a benchmark against which practical coding schemes can be compared.

The performance characteristic we are using relates the relative receiver workload, ζ_R to the relative attack energy, ζ_A. The relative receiver workload is the ratio of the number of complete messages processed by the receiver while under attack to the number processed otherwise; thus

$$\begin{aligned} \zeta_R &= \frac{M_S + M_A + M_H}{M_S} \\ &= 1 + \frac{M_A}{M_S} + \frac{M_H}{M_S} \end{aligned} \tag{F.1}$$

where M_S is the number of messages transmitted concurrently by the legitimate transmitter, M_A is the number of rogue messages transmitted by the attacker, and M_H

is the number of hallucinations produced by the packet. Note that we are assuming that any hallucinations are the result of the attack packet; while not guaranteed, it is reasonable to assume that any encoding scheme that routinely produces hallucinations while not under attack would be be considered acceptable in a practical implementation.

The relative attack energy is the ratio of the number of marks transmitted by the attacker to those transmitted by the legitimate sender, which is equivalent to the ratio of the transmitted packet densities, namely

$$\zeta_A = \frac{\mu_A}{\mu_S} \tag{F.2}$$

Note that this ratio is *not* the same as the ratio of the number of attack messages to the number of legitimate messages. In general, the attack packet density will be sufficient to result in a significant number of mark collisions, thus permitting more messages to be loaded into the packet per mark than than the legitimate sender.

We need to express M_A and M_H in terms of ζ_A and the codec configurations parameters, L, F, eta, and M_S. First, we'll look at M_A and then turn our attention to M_H. The number of messages in the attack packet (recall that we are assuming that the attack packet is generated by superimposing M_A complete messages in order to achieve an attack packet density of μ_A) is directly related to the number of independent marks, N_A, that the attacker can use and is

$$M_A = \frac{\eta N_A}{L} \tag{F.3}$$

The number of independent marks and the corresponding codeword/packet density is given by (7.5) as

$$\mu = 1 - \left(1 - \frac{1}{C}\right)^N \tag{F.4}$$

where C, the number of bits in a codeword, is equal to LF. Inverting this to determine the number of marks as a function of the packet density yields

$$N = \frac{\ln(1-\mu)}{\ln\left(1-\frac{1}{LF}\right)} \tag{F.5}$$

We therefore have

$$M_A = \frac{\eta N_A}{L} \tag{F.6}$$

$$N_A = \frac{\ln(1-\mu_A)}{\ln\left(1-\frac{1}{LF}\right)} \tag{F.7}$$

$$\mu_A = \zeta_A \mu_S \tag{F.8}$$

$$\mu_S = 1 - \left(1 - \frac{1}{LF}\right)^{N_S} \tag{F.9}$$

$$N_S = \frac{M_S L}{\eta} \tag{F.10}$$

and bringing all of these together yields

$$M_A = \left(\frac{\eta}{L}\right) \frac{\ln\left(1 - \zeta_A\left[1 - \left(1 - \frac{1}{LF}\right)^{\frac{M_S L}{\eta}}\right]\right)}{\ln\left(1-\frac{1}{LF}\right)} \tag{F.11}$$

The expression for M_H can be obtained by noting that, for a randomly picked message, the probability that its codeword will be completely covered by the packet will be

$$\Pr(\text{M is covered}) = \mu^N \tag{F.12}$$

where μ is the packet density and N is the number of marks in the codeword. We will assume that the received packet density is the same as the attack packet density – meaning that we assume that the attack packet density is much greater than the density of the legitimate packet. As long as the legitimate packet density is only a few percent or less, the impact of this assumption is negligible. Proceeding with this

assumption and noting that there are 2^L codewords in the codebook, the number of hallucinations is

$$M_H = 2^L \mu_A^{\left(\frac{L}{\eta}\right)} \tag{F.13}$$

combined with prior results, this yields

$$\begin{aligned} M_H &= 2^L \left(\zeta_A \mu_S\right)^{\frac{L}{\eta}} \\ &= \left(2 \left(\zeta_A \mu_S\right)^{\frac{1}{\eta}}\right)^L \\ &= \left(2 \left(\zeta_A \left[1 - \left(1 - \frac{1}{LF}\right)^{\frac{M_S L}{\eta}}\right]\right)^{\frac{1}{\eta}}\right)^L \end{aligned} \tag{F.14}$$

Bringing everything together and recalling that we want

$$\begin{aligned} \zeta_R &= \frac{M_S + M_A + M_H}{M_S} \\ &= 1 + \left(\frac{1}{M_S}\right)\left(M_A + M_H\right) \end{aligned} \tag{F.15}$$

we have

$$\begin{aligned} \zeta_R \;=\; & 1 + \left(\frac{1}{M_S}\right)\left(\frac{\eta}{L}\right) \frac{\ln\left(1 - \zeta_A \left[1 - \left(1 - \frac{1}{LF}\right)^{\frac{M_S L}{\eta}}\right]\right)}{\ln\left(1 - \frac{1}{LF}\right)} + \\ & \left(\frac{1}{M_S}\right)\left(2 \left(\zeta_A \left[1 - \left(1 - \frac{1}{LF}\right)^{\frac{M_S L}{\eta}}\right]\right)^{\frac{1}{\eta}}\right)^L \end{aligned} \tag{F.16}$$

The characteristic is shown for several values of F in Figure F.1. Each curve is plotted with $L = 1000$, $\eta = 1$, and $M_S = 1$. While certainly not obvious from (F.16), the value of L has negligible effect once it becomes greater than a few hundred bits.

Close examination of the curves in Figure F shows the following general behavior: The curves lie essentially on top of the characteristic for a one-hot codec until just shortly before they go singular, at which point they begin a very mild rise above that

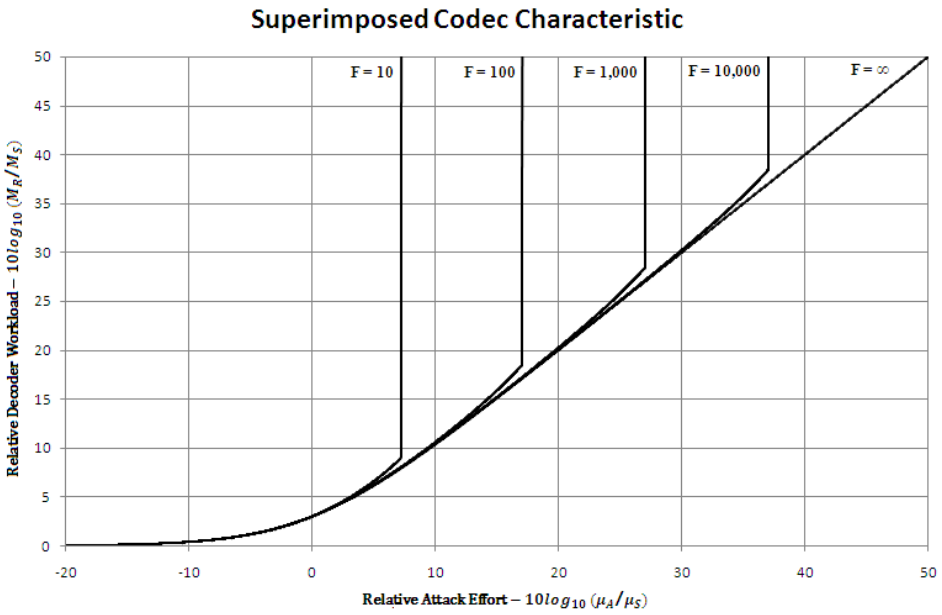

Figure F.1: Generic Superimposed Codec Characteristic.

bounding curve. This rise is due to the increase in the number of attack messages that can supported in a packet on a per-mark basis as the density grows and mark collisions occur with greater frequency. The vertical wall that each curve (other than the one-hot curve) terminates in is due to the explosion of hallucinations once a density of 50% is reached. Prior to that density, there are practically no hallucinations to speak of since they are suppressed exponentially while above that density they grow exponentially.