JAM-RESISTANT COMMUNICATION WITHOUT SHARED SECRETS THROUGH THE USE OF CONCURRENT CODES

LEEMON C. BAIRD III, WILLIAM L. BAHN, AND MICHAEL D. COLLINS

U.S. Air Force Academy Technical Report
USAFA-TR-2007-01
14 February 2007
Distribution A, Approved for public release, distribution unlimited

Abstract. We consider the problem of establishing jam-resistant, wireless, omnidirectional communication channels when there is no initial shared secret. No existing system achieves this. We propose a general algorithm for this problem, the BBC algorithm, and give several instantiations of it. We develop and analyze this algorithm within the framework of a new type of code, *concurrent codes*, which are those superimposed codes that allow efficient decoding. Finally, we propose the Universal Concurrent Code algorithm, and prove that it covers all possible concurrent codes, and give connections between its theory and that of monotone Boolean functions.

Contents

This work was sponsored in part by the Air Force Information Operations Center (AFIOC), Lackland AFB, TX, and was performed at the Academy Center for Information Security (ACIS) at the United States Air Force Academy.

1. Introduction

Jam resistance is becoming increasingly important to modern communication. It is increasingly important to the military, as operations become more dependent on real-time, wireless communication, and even brief denial of service attacks become more damaging. It is increasingly important in the civilian sector, now that cheap, tiny cell phone jammers are becoming available. It is becoming more important for civilian GPS signals, because they are used by airlines and others, so jamming could cause serious problems. As the world moves toward software defined radios, the tools to jam police and emergency frequencies will become increasingly widespread. Jam resistance is a critical issue.

Traditionally, jam resistance has been achieved through spread spectrum communication. Each of the three common forms of spread spectrum can be made resistant to jamming if the sender and receiver share a secret key. Much attention has been given to performance against jamming and mitigation techniques in spread spectrum systems both generally [1, 2, 3] and in specific subfamilies including direct sequence [4, 5, 6, 7, 8, 9, 10], frequency hopping [4, 11], and pulse-position systems [12, 13, 14].

In *frequency hopping*, the sender and receiver communicate on a single frequency for a short period, then jump to another frequency. Bluetooth uses frequency hopping, and changes frequencies every 0.625 ms. If the sequence of frequencies is a cryptographically-secure, pseudorandom sequence determined by a secret key, then an attacker will not know which frequency to jam at any given time. The attacker must therefore flood every possible frequency with energy to guarantee that every part of the message is destroyed. This requires an enormous amount of energy. A smart jammer can usually succeed by destroying only a portion of the message, which requires fewer frequencies to be jammed, and so requires less energy. But it still requires the attacker to expend far more energy than the legitimate users. This can be expensive, can require large devices, and can make it easier to detect and triangulate the location of the attacker. This asymmetry in energy usage is the standard goal for a jam-resistant wireless communication system.

In *direct sequence*, a sender uses all frequencies simultaneously, by combining the message with a pseudorandom bitstream, generated according to some key. This is used in CDMA (Code Division Multiple Access) cell phones [15, 10, 16, 17, 9, 18]. If the key is known, then the attacker can predict the sequence, and can broadcast a strong signal using that sequence, thus masking the signal sent by the legitimate user. If the key is secret and a strong sequence is used, an attacker is forced to expend far more energy than the legitimate sender. A related approach is Orthogonal Frequency Division Multiplexing (OFDM) CDMA spread spectrum [19, 20] which uses multiple, orthogonal frequencies simultaneously.

In *pulse-based* systems, the sender broadcasts a series of very short pulses, each of which essentially spreads noise over the entire spectrum. The message is encoded in the exact timing of the pulses, and this encoding can be dependent upon a secret key if jam resistance is desired. This system is used in many of the new Ultra Wide Band (UWB) systems that are in the early stages of development and deployment [21, 22, 23, 24, 25, 26, 27, 28]. Although the pulse contains a fairly small amount of total energy, it is so short that the power is very high. This makes it difficult for a jammer to mask the pulse without using an enormous amount of energy to broadcast continuously. If the key is known, then the attacker can easily send pulses

corresponding to the zeros in the message, and the receiver will receive a message consisting entirely of 1 bits. If the key is unknown, then the attacker must resort to sending pulses during every time period to destroy all of the message. Again, a smart jammer can succeed by only destroying a portion of the message, but it can still require far more energy than is used by the legitimate sender.

In each of these three systems, it is possible for multiple users to transmit without jamming each other, as long as each user is polite and uses a different channel (a different seed for the pseudorandom generator). However, an attacker will not be polite, and can easily jam the system if the channel is known. In each of these three systems, jam resistance has only been possible by using a secret key to control the sequence of frequencies, or the chip sequence, or the timing of the pulses. If the secret is known, then an attacker can concentrate energy in the channel being used (defined by the key), and easily jam the communication with little energy expenditure. That is why the channel must remain a secret.

Unfortunately, secret keys are not scalable to large systems. If the civilian GPS signal used a key, then that key would have to be distributed to all 6 billion potential users, including the attackers. There is no way to use a secret key to protect such a signal that is meant to be public.

Handheld jammers for cell phones can now be bought on the open market for under $200 [29]. Theoretically, cell phones could be made jam resistant by using a separate secret key for each customer. But then each cell phone tower would have to store the secret keys for every customer in the world who might enter that cell. The tower would also have to continuously monitor all of those millions of channels. The cell phone system is at least based on fixed locations, but the increasing use of ad-hoc networks in sensitive and mission-critical roles raises significant issues in how those networks will be protected against jamming and other malicious attacks [30, 31, 32, 33, 34, 35].

The scaling problem is even worse for military applications. The future of the US Air Force is said to be net-centric, joint, and coalition. The current vision is that the entire battlefield will be a single ad-hoc wireless network, with continuous packet forwarding between devices owned by different services, and even by different countries. That would require that a single secret key be used by all of the equipment in the entire theater. If an enemy captured even a single handheld radio or micro-UAV, it would allow jamming of wireless communication throughout the entire the theater until new keys could be reloaded.

The current situation is analogous to the state of cryptography in the early 1970s. Symmetric ciphers based on secret keys were believed to be secure, but the infrastructure for key management did not scale well. The invention of asymmetric cryptography [36, 37, 38, 39] allowed the development of a Public Key Infrastructure (PKI) which did scale well and allowed efficient key distribution [40, 41, 42, 43]. There is now a critical need for the equivalent of PKI for jam resistance.

Specifically, there needs to be some way that two strangers with no shared secret can exchange messages over a wireless medium in such a way that an attacker positioned between them cannot easily prevent those messages from arriving. Ideally, the attacker should be able to inject new messages into the stream, but not block or modify any legitimate messages. Attacks on traditional shared-secret jam-resistant communication require the attacker to expend far more energy than the legitimate user, flooding the entire spectrum with energy. This can be difficult for the attacker

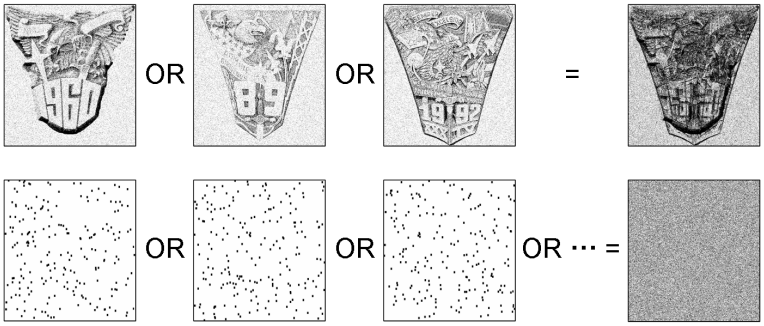

FIGURE 1. A visual example of the decoding problem. The human eye cannot pull apart 3 pictures that have been superimposed with a bitwise OR (top), yet BBC successfully decodes a combination of 1000 images (below) without making any errors whatsoever. In the bottom example, each image (codeword) has 480,000 pixels (bits), of which about 208 are black (1), which encodes a 200-bit message (using $k = 8$).

to achieve, and it makes it easier to locate and deal with the attacker. The need now is for an equivalent level of jam resistance without the shared secret.

No existing system achieves this goal. One might imagine building a system based on error correcting codes, but existing error correcting codes cannot even decode two messages that have been sent simultaneously and are combined with a bitwise OR. Various other codes such as locally decodable codes [44] or X codes [45] or separating codes [46] appear to be even less useful in this case. Superimposed codes deal with that situation, but traditionally have not allowed efficient decoding. If many messages are encoded and then combined with a bitwise OR, it generally appears difficult to recover all of them, especially when the space of possible messages is exponentially large. The difficulty of this problem is illustrated in figure 1.

This paper proposes the first system for jam-resistant, omnidirectional, wireless communication without a shared secret. It works by defining codewords that can be combined with a bitwise OR, and then reliably and efficiently decoded. Related work in coding theory will be presented first, followed by the general BBC algorithm, and then instantiations for each of the three major types of spread spectrum. Finally, additional mathematical theory and other applications of this work will be discussed.

2. Related Work on Codes

A great deal of research has been done related to binary codewords that are combined with a bitwise OR, dating back at least as far as 1956 [47, 48], and being formalized as early as 1964 [49]. This progress has been made in a wide variety of fields, often with only a small amount of contact between them. Work has been done in information retrieval, coding theory, and at least three different branches of mathematics according to one survey. This has led to a large array of terms for the same concepts.

One binary string is said to be *contained* within another string of the same length if every 1 bit in the first string is in a position where the second string has a 1. In that case, it is also said that the first string is a *subset* of the other. We will use the former term in this paper.

If several codewords are combined with a bitwise OR, it is desirable that no other codeword should be contained within that combination. Codes to achieve this in all (or most) cases are variously known as *superimposed codes* [49, 50, 15, 51], *Bloom filters* [52, 53], *strongly selective families* [54], and *cover-free families* (or *r-cover-free families*) [55, 56, 57, 58, 59]. These general terms are all roughly synonymous, though many papers extend and constrain them in various ways, to get categories such as (s,l)-superimposed codes or other generalizations [60, 61, 62, 63, 64, 65, 66, 67, 68]. Typically, these codes a fairly small codebook, such as one codebook per user, so it's more likely to be on the order of 1000 codewords rather than 2^{1000} [69, 70, 71, 72, 73]. In this paper we will not consider this set (by any of these names), but will instead propose and analyze a new subset of them, the *concurrent codes*.

If several codewords are combined, there may be other codewords that are accidentally contained in the result. In this paper these will be called *hallucinations*, because they are unintended messages that the receiver sees that were never intentionally sent by any sender. Other terms for these include *false drops*, from the old practice of inserting rods into notched punch cards and seeing which cards drop out, *foreign codewords*, because they are foreign to the set of codewords that were included intentionally, *false covering errors* [74], because the packet covers the unwanted codewords, and *false positives*, from the presumed question "is the given codeword included?"

In this paper, we call several *codewords* that are combined with bitwise OR a *packet*. Other terms in common use include *subsets* that are combined via set *union* to yield a *set*, or *signatures* that are combined via *disjunction* to yield other *signatures* (or to yield a *Bloom filter*), or *bit vectors* that are combined via *addition* or *sum* [75] to yield other *vectors*. This usage of "addition" is particularly unfortunate, since a bitwise OR (disjunction) is not the same as addition over GF(2) or any other field or group. Yet many papers call the operation addition, and use the standard symbols associated with that operation.

A channel where multiple senders have their broadcasts combined in this way is widely called a *multiple access OR channel* [74] or *multiaccess OR channel* [76], or simply the *OR channel* [77].

An enormous amount of work has been done to design codes that ensure there will be no hallucinations when up to some fixed number of codewords are combined into a packet. Some work has also been done on randomized codes that simply have low probabilities of hallucinations in that case. For the most part, such work is unrelated to the work in this paper, though our analysis of random codebooks in section 10 is almost identical to analysis that has appeared in many different papers through the years.

The US National Institute of Standards and Technology (NIST) defines a superimposed code as "A set of bit vectors such that no vector is a subset of a bitwise OR of a small number of others." [78] Others tend to use the term in a slightly more general sense, allowing it to also include codes that usually (rather than always) have no such vector, considering the average case rather than just the worst case.

We define *concurrent codes* to be a subset of that more general definition. A concurrent code is defined to be a superimposed code that can be decoded in polynomial time. This is formally defined in section 12. The computational complexity should be a polynomial function of the number of bits in a codeword and the number of messages that are combined in a packet, not a function of the size of the codebook. So if there are 1000 codewords in a packet, chosen randomly from a set of 2^{1000} codewords in the codebook, it should still be possible to recover the codewords quickly, despite the codebook itself being exponentially large. This definition allows the decoding time to be a large polynomial, even though all the algorithms we propose run in linear time.

There has traditionally been very little attention paid to efficient decoding of these codes. It is often assumed that the codebook will be small, with perhaps one codeword per user (in a communication system) or per word (in a dictionary) or per document (in an information retrieval system). In all these cases, the number of codewords in the entire codebook is likely to be closer to 1000 than to 2^{1000}. In this case, complete decoding is easy: simply compare every codeword in the codebook to the packet. If the bitwise OR of a codeword with a packet equals that packet, then the codeword is contained in that packet. Furthermore, in most traditional contexts, there is no need to completely decode a packet. The more common question is whether a particular, known codeword is contained in the packet. That question is quickly answered.

An algorithm was published in 1988 for decoding a packet of n messages in n^3 or n^4 time, for packets containing fairly small numbers of messages [79, 76, 80, 81, 82]. So a packet with 1000 messages could require on the order of a trillion correlations to be calculated. A paper in 1995 repeated that algorithm and said it could be extended to handle noise, though the extension was not given in the paper for the sake of "simplicity" [83]. Other than these two brief results, we are aware of no other work related to concurrent codes. That is probably due to their lack of usefulness for the problems traditionally addressed.

We will show, however, that concurrent codes are actually very useful. They can allow jam-resistant communication with no initial shared secret. Furthermore, we will show that all possible concurrent codes can be viewed as special cases of the Universal Concurrent Code, and will prove several theorems about that relationship, and their relationship to monotone Boolean functions and monotone Boolean circuits. This suggests the existence of an interesting underlying theory for concurrent codes, which we have only started to develop. The simplest concurrent code that we have found is the BBC algorithm, which is described in the next section.

3. The BBC Algorithm

The general BBC algorithm is given in Algorithm 1 (for sending) and 2 (for receiving). The broadcast algorithm tells how to encode and send a single message. The decoding algorithm assumes that multiple, encoded messages were sent simultaneously, combining their binary strings with a bitwise OR, to form a combined string known as a *packet*. In addition, random noise (or the results of an active attack) will flip some of the packet's bits from 0 to 1 (but never from 1 to 0). Under these assumptions, the packet will have the same number of bits as a codeword, but typically many more of the bits will be 1 in the packet than in a single codeword.

The constant k controls the number of checksum bits to use (the bits themselves are simple zeros, but they force the hash function to create a strong checksum). The hash function H maps an arbitrary-length bit string to a location. The general BBC algorithm requires the making of *indelible marks* at *locations* chosen by hashes of all prefixes of the message to be sent. The exact definition of *indelible mark* and *location* depends on the system being used, but the requirement is that both the sender and attacker can make such marks, but neither can erase them. So it is equivalent to a long string of bits that are initially all zero, and the sender and attacker can each change any 0 to 1, but cannot change a 1 to 0. The attacker could jam the communication by setting all the bits to 1, but if there is an energy cost associated with each 1 bit, and if the legitimate sender is setting only a thousandth or a millionth of the bits to 1, then the attacker will be forced to expend a thousand or million times more energy than the sender.

Algorithm 1 BBCbroadcast(M)

This function broadcasts an m-bit message $M[1 \ldots m]$ adding k checksum bits to the end of the message. H is a hash function. The definition of H and the values of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

Append k zero bits to the end of M
for $i \leftarrow 1 \ldots m + k$ **do**
  Make an *indelible mark* at the *location* given by $H(M[1 \ldots i])$
end for

Perhaps the simplest instantiation of BBC is when pulses are used, as in the most recent Ultra Wide Band (UWB) systems. The sender broadcasts very short, high-power bursts of radio frequency noise at certain times. The message is encoded in the timing. An attacker can also broadcast such pulses, but cannot erase any existing pulses. Such pulses are difficult to erase because they are very short, very high power, and consist of unpredictable random noise with energy spread over the entire spectrum. No known system exists that can detect and analyze such pulses, and then send out an inverse waveform to cancel them. There simply isn't time during the brief period it takes for the pulse to pass the attacker.

Figure 2 shows how to broadcast using *BBC-Pulse*, an implementation of BBC for pulse-based broadcast. In this example, the message $\mathbf{M = 1011}$ is padded with $k = 2$ zero bits to get 101100. All prefixes of this are hashed with the hash function defined by the table on the right. A pulse is sent at the time defined by the hash of each prefix of 101100. For example, $H(1) = 21$, so a pulse is sent at time 21, and $H(10) = 9$, so another pulse is sent at time 9.

Figure 3 shows the decoding of that same message for BBC-pulse. The receiver starts at the root of the tree (on the far left), which is an empty string. The receiver then repeatedly tries appending both a 0 and 1 bit to each string being considered, and calculates the hash of each new string generated. So on the first round, the receiver calculates H(0)=4 and H(1)=21. Since a pulse was not received at time 4, the receiver knows that no messages starting with 0 could have been sent. Since a pulse was received at time 1, the receiver knows that at least one message was

Algorithm 2 BBCdecode(n)

This recursive function can be used to decode all the messages found in a given packet by calling BBCdecode(1). There must be a global $M[1 \ldots m+k]$ which is a string of $m+k$ bits. The number of bits in a message is m, and the number of checksum zeros appended to the message is k. H is a hash function. The definition of H and values of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

if $n = m + k + 1$ **then**
  output "One of the messages is:" $M[1 \ldots m]$
else
  if $n > m$ **then**
    $limit \leftarrow 0$
  else
    $limit \leftarrow 1$
  end if
  for $i \leftarrow 0 \ldots limit$ **do**
    $M[n] \leftarrow i$
    if there is an *indelible mark* at *location* $H(M[1 \ldots n])$ **then**
      BBCdecode(M,$n+1$)
    end if
  end for
end if

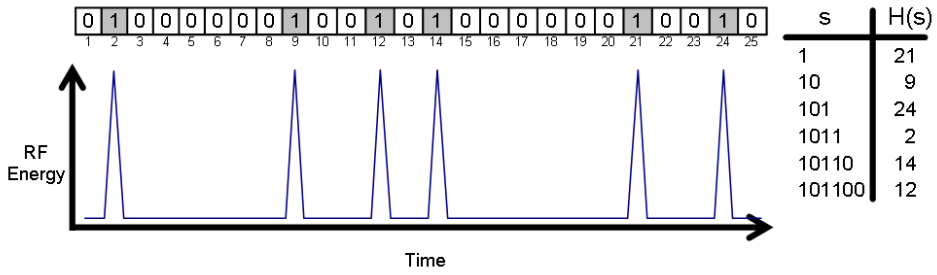

FIGURE 2. BBC broadcast using UWB pulses. $M = 1011$, $k = 2$, an *indelible mark* is a radio pulse, and its *location* is the time when the pulse occurs relative to the start of the message. The table on the right shows part of the definition of the hash function $H(x)$.

sent that started with 1. These results are shown on the tree by coloring the 0 box white and the 1 box gray.

This process then continues. The string 0 is not expanded because no pulse was detected at time H(0). The string 1 is expanded both ways to get 10 and 11, and the receiver calculates H(10) and H(11). Both of these hashes yield times at which pulses were received, so both boxes are colored gray, and both will be expanded on

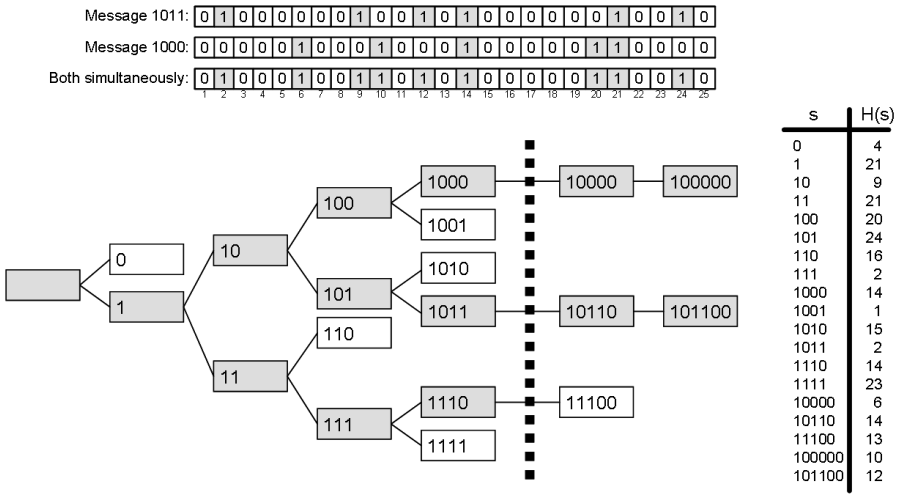

FIGURE 3. Decoding tree for *BBC-Pulse* broadcasts. In this example, both of the messages 1011 and 1000 were broadcast simultaneously, and $k = 2$. The table on the right shows part of the definition of the hash function $H(x)$. The 3rd row at the top shows at which times a pulse was sent (gray boxes) or not sent (white). This is essentially a bitwise OR of the pulse pattern for each message. The binary tree shows those prefixes that are considered during decoding. A prefix is gray if its hash gives a time at which a pulse was detected. The dotted line shows the transition from decoding the body of the message to checking the appended checksum zeros.

the next round. This process continues until the entire 4-bit message has been decoded (at the dotted line). Note that at this point, both of the legitimate messages have been found (1011 and 1000), but an additional "message" has also been found: 1110. This third message is a *hallucination*. This hallucination occurred because all three of the strings 11, 111, and 1110 hashed to locations that just happened to contain a pulse for other reasons. This is why the k checksum bits are important. To the left of the dotted line, every box is expanded by appending both 0 and 1. To the right of the line, only 0 bits are appended. This is because every legitimate message is known to have k zero bits at the end. Both of the legitimate messages survive through all k rounds of checking. The hallucination, however, is eliminated because the string 11100 does not hash to a location that happens to contain a pulse.

This system is intended to be used in situations where a very small fraction of the time positions contain pulses. But consider a worst case, where a full third of all positions contain pulses. If a good cryptographic hash function is used, then these pulses will be scattered pseudorandomly. Then during the checksum process, at each step a given hallucination will have a probability of only 1/3 of surviving, because its hash will choose a pseudorandom location that has a probability of only 1/3 of containing a pulse. If there are k checksum bits, then the probability of a

BBC-pulse
mark - a very short pulse of radio frequency noise
location - the time at which the pulse occurs
BBC-frequency-hopping-simultaneous
mark - a pure sinusoidal signal (all marks sent simultaneously)
location - the frequency of the signal
BBC-frequency-hopping-sequential
mark - a pure sinusoidal signal for a short duration
location - $H(x)$=frequency, length(x)=start time
BBC-direct-sequence
mark - a pseudorandom radio frequency waveform
(e.g. a chirp sequence)(all marks sent simultaneously)
location - the seed to use for the waveform generator

FIGURE 4. Instantiations of the BBC algorithm for each of the major spread spectrum techniques

hallucination surviving until the end is only $(1/3)^k$, which quickly becomes vanishingly small for even moderate values of k. The number of potential hallucinations will be small (as is proved in a later section), therefore, the receiver is very unlikely to end up with any hallucinations at all. This, of course, assumes only random noise and accidental interference between legitimate messages. The analysis of an active attack is more complex, and is given in a later section.

All of the above analysis is for the pulse-based instantiation of BBC, but it applies essentially unchanged to the other instantiations as well. Figure 4 lists instantiations of BBC for each of the three major spread spectrum techniques, including both sequential and simultaneous approaches.

The two simultaneous instantiations (BBC-frequency-hopping-simultaneous and BBC-spread-spectrum) have several interesting properties. In BBC-frequency-hopping, the locations are frequencies, and a mark is a pure sine wave broadcast on the appropriate frequency. In the simultaneous version of it, all of the sine waves are broadcast simultaneously. That would be difficult on older hardware that uses physical circuits or crystals for each frequency, but it can be done easily on modern software defined radios, which are ideal for trying new communications schemes and for both attack and defense [84]. It could also be done easily with a single, custom chip, where a separate part of the circuit is devoted to generating each frequency. Either way, it has interesting properties not found in any system currently in use.

This approach sends information *holographically*. In a long broadcast, the entire message is encoded in each short period of time. Theoretically, the receiver can listen for a very short period and capture the entire message at once. In practice, the length of the period will depend on the level of background noise and the quality of the receiving hardware. If there is a high level of noise, and the receiver is a small antenna with low-quality amplifiers etc, then the receiver will need to capture and process data from a long period in order to recover the message. But as soon as the noise drops, or the receiver upgrades to a larger antenna and better hardware, the message will be clear much sooner. This suggests an interesting approach sending messages in a way that adapts to available bandwidth. The sender can simply broadcast a message continuously until an acknowledgement is received. It also

suggests reduced synchronization problems, since the receiver can start receiving in the middle of a message and still recover the entire message. The direct sequence instantiation of BBC also has this property, with an additional property that the receiver will be able to determine the exact instant that the sender must have started the broadcast, even if the receiver started receiving slightly later. This would be useful for time-based systems such as GPS.

4. BBC Demonstration

A demonstration of the BBC algorithm was written by two colleagues, Martin Carlisle and Sean Butler. It allowed multiple computers to broadcast short messages using BBC, and another computer to receive and decode those messages. For others interested in replicating this demonstration, we give here the specific parameters used.

The demonstration was designed to use all of the algorithms and processing that would be used in a real-world system, but to apply them to very short messages (8 bits of data) using sound waves rather than electromagnetic waves. This allows laptops to communicate using their built-in speakers and microphones, without having to add peripherals such as Software Defined Radios (SDR). A demonstration using SDR is now being built, but the sound-based system still has the advantage that it can be run on any commercial laptop without additional hardware. It also has the advantage that human observers can hear the messages passing through the air, whereas they are invisible in an RF-based system.

The system is designed to emulate a pulse-based UWB radio. A pulse is generated as a short click, consisting of a pure tone at 2.812 kHz for a duration of 2 milliseconds. If a pulse is sent in every possible time slot, then there is 2 ms of silence between adjacent pulses. This sounds similar to a geiger counter. The computer's microphone was sampled at 22.5 kHz to get a series of samples x_t. Windows represents each sample as a signed integer in the range $[-659, 620]$. The samples were first passed through this first-order FIR bandpass filter with cutoff frequencies at 2.25 kHz and 5.5 kHz:

$$\begin{aligned} y_t = &- 0.03497393x_t - 0.12694655x_{t-1} + 0.08117938x_{t-2} \\ &+ 0.48028642x_{t-3} + 0.08117938x_{t-4} - 0.12694655x_{t-5} - 0.03497393x_{t-6} \end{aligned} \tag{4.1}$$

which was immediately followed by this simple IIR filter:

$$z_t = 0.9z_{t-1} + y_t^2 \tag{4.2}$$

Then z_t was downsampled by a factor of 5, dropping all but every 5th sample. The receiver then defined the start of a pulse as any time that z_t exceeded 2,000,000, and the end of the pulse as whenever it dropped below 2,000,000. It would be possible to use two different thresholds for rising and falling, to provide a hysteresis effect, but that wasn't found to be necessary in this case.

On the sending computer, the user was allowed to enter text on the keyboard. Each character typed became a separate message, in 7-bit ASCII. An 8th bit was prepended to each message to simulate a signature. One sender always used a 0 and the other always used a 1. The receiving computer displayed a split screen, showing all characters received, separated according to their signatures. In a real

system, of course, a digital signature would need to be much longer than a single bit to be secure.

Each message was encoded as a codeword of 184 bits, of which at most 24 bits were set to 1. The encoding was BBC with the parameters from the top row of figure 5. The codeword was sent as a single pulse followed by 184 time slices (of 4 ms each), with a 2 ms pulse in any slot with a 1, and silence for any slot with a 0. This is the same as if codewords were 185 bits, and every codeword started with a 1.

Because of the small codebook (only 256 possible messages), this could have been decoded using exhaustive search. But that is not scalable, so BBC was used to decode it instead. Every pulse that was received was treated as the first pulse of a packet, and a complete decoding was performed based on that assumption. In most cases, the decoding tree died out almost immediately, so very little computation was wasted. For pulses that actually were the starts of legitimate packets, the decoding tree yielded the actual message, and it was displayed on one of two sections of the receiver's screen, depending on the signature bit. If an identical message was received twice, with the same signature, with one starting within 144 (downsampled) samples of each other, then the second copy was ignored.

The demonstration ran with all 3 laptops sitting adjacent on a desktop, with the microphone of the receiver equidistant from the speakers of the other two. The system appeared to be reliable, even in the presence of background noise, and even when both senders were sending simultaneously. This system also worked well when the sending and receiving programs were run simultaneously on a single computer, because the laptop's microphone was close enough to its speaker.

A new system is now being built using software defined radios. But the majority of the software will remain unchanged, and the audio system will continue to serve as a useful demonstration.

5. Hybrid Communication Systems

Traditionally, public key cryptography (or *asymmetric* cryptography)is more powerful than symmetric cryptography, in the sense that it allows strangers to communicate securely without having previously established a shared key. However, the fastest known public key ciphers are far slower than the best symmetric ciphers. Therefore, cryptographers typically build hybrid systems. When Alice wants to talk to Bob, she first generates a random session key. She then encrypts this with Bob's public key and sends it to him. She might even digitally sign it with her own private key, to prove it comes from her. All this uses a slow, asymmetric cipher, but that is acceptable, since there are only a few hundred bytes to be sent. Once Bob has decrypted the session key, all future communication is done using that session key with the symmetric cipher, which is far faster.

A similar hybrid system should be used for jam-resistant communication, for much the same reasons. If Alice wants to communicate with a stranger Bob, she would first generate a random session key, encrypt it with Bob's public key, sign it with her private key, and send it to him in a jam-resistant way using BBC. The BBC communication may be slightly slower than traditional spread spectrum for any given degree of jam resistance (about half the speed) and it requires more computation. This is perfectly acceptable, since only a few hundred bytes are sent.

Once Bob has received the message from Alice (and perhaps other messages from the attacker), he then decrypts them with his private key, and checks their signatures. The BBC algorithm ensures that this single message will arrive intact, and be resistant to jamming. The encryption with Bob's public key ensures the attacker will not know the session key. The signature with Alice's private key ensures that the attacker cannot impersonate Alice. At this point, Alice and Bob now share a secret, random, session key, and can communicate from then on using traditional spread spectrum using that session key. Since it is traditional spread spectrum, it will make efficient use of the available bandwidth, and messages can be sent quickly. Since the session key is unknown to the attacker, the channel will be resistant to jamming.

Therefore, this hybrid system for jam resistance is exactly analogous to hybrid systems for encryption. Just as the invention of asymmetric ciphers made secure communication between strangers practical, so BBC makes jam-resistant communication between strangers practical. In both cases, the new cryptographic primitive allows entirely new protocols that greatly reduce the difficulty of scaling up security to large populations of people.

In fact, BBC can even use the existing Public Key Infrastructure (PKI). For example, the U.S. military is in the process of giving every member a smart card that contains public and private keys. If two DoD members need to communicate in a way that is both secure and resistant to jamming, they could each insert their smart cards into their BBC-enabled radios. The two radios could then use BBC to establish a session key, then use traditional spread spectrum to communicate thereafter. This leverages the existing PKI infrastructure, and means there is no need to set up a special jam-resistance infrastructure.

In some cases, it may not be necessary to use a hybrid system, and BBC can be used directly. The best example of that is the civilian GPS system described in the next section.

There is a further question of interest: if arbitrarily-large amounts of data are to be sent with jam resistance, broken up into small packets, what is the absolute smallest packet size that would still give security? For most applications, including the communication discussed in this section and the civilian GPS discussed in the next section, it will be necessary to digitally sign each message. Currently the most efficient digital signatures are based on elliptic curve cryptography. The National Security Agency (NSA) Suite B standard calls for use of elliptic curve Digital Signature Standard (DSS) 256-bit keys for signing secret data, and 384-bit keys for signing top secret data. The signature is twice the size of the key, so the signatures would be 512 and 768 bits respectively. To be conservative, assume 768 bits for the signature.

In addition to the signature, a minimal-sized message would include the current time and date to preclude replay attacks. The number of nanoseconds in a century is less than 2^{62}, so the timestamp would require at most 62 bits, as a conservative estimate.

If each message has both a timestamp and signature, then theoretically, arbitrary amounts of data could be sent with just 1 bit of useful data per message. That gives a total message size of $768 + 62 + 1 = 831$ bits. Given enough messages, this could be used to send the identity of the sender, PKI certificates, and actual data. If the payload is expanded from 1 bit to 194 bits, that information could be

transferred relatively quickly, and the message size would expand to 1 kilobit. If a spread spectrum system has the ability to send a gigabit per second, and this kilobit is sent in one second, then only one millionth of the available bitrate will be used. For traditional spread spectrum (with shared secrets) this would give a *processing gain* of one million, which would require the attacker to expend on the order of one million times the energy as the sender in order to jam the transmission. As will be shown below, if BBC takes twice as long to transmit, then it can also achieve that factor of a million, and so it would have approximately the same security as traditional spread spectrum, with no shared secrets, and with approximately half the data rate. For a hybrid system, this halving of the data rate would be very acceptable, because only one message (or a few) would need to be sent before switching to traditional spread spectrum.

6. Public Broadcast - The Civilian Global Positioning System

The Global Positioning System (GPS) was originally designed for military use, and attempted to achieve jam resistance through the use of traditional spread spectrum with a secret key. The satellites continuously broadcast the current time. The user compares those times and uses their differences to estimate distances to the satellites. Given several such distances, the user's location can be found.

A civilian channel was also implemented, which was not encrypted and did not have any kind of jam resistance at all. As it turned out, the civilian channel was extremely useful. An entire industry sprung up around it, and it is now used in critical applications such as the airline industry. However, this raises a disturbing possibility. The civilian channel cannot use a secret key, by definition, since it is intended to be usable by every person on earth, including the attackers. There was no known way to make it resistant to jamming. Because of its success, that is now a serious problem. How can we prevent terrorists from jamming civilian GPS, when a shared secret is literally impossible?

The solution is to use BBC directly, rather than in a hybrid mode. Although any form of BBC can be used (except *BBC-frequency-hopping-simultaneous*, it would seem most reasonable to use either *BBC-pulse* or *BBC-direct-sequence*. The GPS satellite composes a message giving the current time (plus some of the almanac and ephemeris data), and digitally signs it with a private key. It then broadcasts this message using BBC. A receiver on the ground decodes the message, and if the signature is valid, assumes that the time referred to the beginning of the message (either the first pulse in *BBC-pulse*, or the start of the waveform in *BBC-direct-sequence*). An attacker cannot forge the message, because of the signature. At best, an attacker can simply record the message and replay it (or detect each pulse or waveform from the satellite and replay it immediately). The user should then use the first copy of the message received, and ignore all later copies. If the message was received multiple times, the user associates the time with the copy. The only remaining attack is for the attacker to send many false messages, hoping to consume all of the user's processing power. However, this can require substantial power on the attacker's part, so the system is as jam-resistant as traditional spread spectrum, even thought the users lack a shared secret.

7. Indelibility

The general BBC algorithm depends on marks being *indelible*. An attacker can create new marks, but must be unable to delete existing marks.

In practice, most forms of broadcast already have this property. For example, a simple Amplitude Modulation (AM) signal is easy to jam by broadcasting noise at that frequency. That prevents the receiver from obtaining useful information from the signal, but doesn't hide fact that there is energy being transmitted at that frequency. To completely hide even the *existence* of energy at that frequency, the attacker must be far more sophisticated. The inverse of the sine wave must be sent to cancel out the original signal. The inverse signal must be exactly the same amplitude, and it must be kept exactly out of phase. The attacker will need to be located along the straight line between sender and receiver, or must know their locations exactly and compensate for the different distances. If there are multi-path reflections, the attacker will need to cancel each of them independently. If the sender's phase drifts slowly over time, then the attacker must track it. For all these reasons, most radio signals are practically indelible.

However, it is useful to make the signals indelible in theory as well as in practice. The pulse-based system already achieves that, assuming the pulse is pure noise rather than some predictable waveform. The frequency-hopping system can achieve indelibility by adding additional randomness. Instead of hopping between frequencies, it should hop between frequency bands. While in a given band, the sender can vary the frequency randomly within that band, as well as randomly change the phase periodically. The receiver need only detect the presence of energy anywhere in the band, so this will still work. The attacker will be unable to predict the exact frequency and phase at any given time, and so can't completely eliminate the signal, even in theory.

Similarly, a direct-sequence system can be made indelible, even in theory. In this system the waveform is pseudorandom. The most common implementation is to have a fast chip sequence (pseudorandom square wave) which is XORed with a slower message bit sequence. Theoretically, an attacker who knows the pseudo-random sequence could hide even its existence by broadcasting the inverse of the signal (though this would be *very* hard to achieve in practice). However, this can be prevented, even in theory, by multiplying the signal with truly-random multiplicative noise which on average reduces the power of the signal by 1/2. If the broadcasted waveform (in the time domain) is the product of the pseudorandom sequence with truly-random noise, then the attacker cannot eliminate the signal completely because it cannot be predicted at each instant. And the receiver can still detect the signal by convolution with the pseudorandom sequence, because the noise was multiplicative rather than additive.

In each case, indelibility is achieved by combining known pseudorandomness with unknown true randomness. This is a nice theoretical result, but probably isn't worth bothering with in practice. Although it is easy to jam a message sent on a known channel, it is difficult to hide the fact that *something* is being sent. Almost any signal will be indelible in a practical sense. Still, it's encouraging that indelibility can also be achieved in theory.

8. Timing and Synchronization

Timing and synchronization are critical for wireless communication, and so attacks on synchronization should be considered for jam resistance. For the simultaneous, frequency-hopping version of BBC, there are no timing or synchronization issues. The sender simply broadcasts energy on a set of frequencies for a long period, and the receiver can listen to any portion of that and recover the entire message. Similarly, the direct sequence version of BBC has little problem with timing issues because all of the indelible marks are sent simultaneously. The receiver does have to lock on to the waveform, but this is exactly the same problem as when current direct-sequence radios have to synchronize with the sender's chip sequence.

However, the pulse version of BBC does have some interesting synchronization issues. If the receiver doesn't know when the message might be sent, then the sender should send a single pulse at the start of each packet. The receiver can then decode the message by looking for pulses at times relative to that first pulse. If there are multiple senders, all out of synch with one another, then the receiver will need to treat every pulse received as the start of a packet. Of course, when most of those pulses are decoded, the decoding tree will quickly die out, and the receiver will conclude that no message was sent at that time. In this approach, an attacker can send a message at any time, and it will never interfere with the legitimate sender.

In that approach, each pulse *location* is a short window in time relative to the initial pulse. If the sender and receiver are both expected to have high-quality equipment with low clock skew, then each window can be very short, and so a long message can be sent very quickly. If the system must run on lower-quality hardware, then the windows must be made longer. Typically, it will be best to define windows as being shorter near the start of the packet and longer near the end. That is because the clock will have had less time to drift near the beginning.

This idea of synchronizing only at the start of the packet is simple, and can be done with cheap hardware, but it may end up taking longer to send a message. The bit rate can be increased tremendously through the use of special hardware. For example, there now exist atomic clocks that are the size of a grain of rice, and which are accurate to one part in 10^{10}. These chip-scale atomic clocks [85] offer the potential to place oscillators that are three to four orders of magnitude more accurate and precise than present crystal oscillators in battery-powered handheld equipment. Radios using such clocks would have extremely low clock drift, and so there would be almost no clock skew between the sender and receiver. This would allow the windows to be extremely short, packing the bits together densely, and allowing a message of a given size to be sent very quickly.

However, it would be useful to achieve high bit rates using software rather than hardware. This can be done with a self-synchronizing system, using every pulse to resynchronize the sender and receiver. The receiver's clock would be adjusted to synchronize with the pulse at the start of the packet. This would then predict two windows in time where the second pulse might be expected. Once a second pulse is found in those windows, the receiver would use the exact time of that pulse to adjust predictions of when the third pulse would be expected.

In this way, the decoding tree would be augmented with exact times for each pulse, and so the estimates could be improved. In fact, the synchronization would become more accurate as the decoder worked its way deeper into the tree. When

the 100th pulse is being decoded, the receiver can look at the 99 pulses already interpreted as part of this codeword, and find the 2 pulses that are closest in time to the 2 locations where the 100th pulse is expected. If they are close, then the window can be made very short for the 100th pulse, since the sender's clock could not have drifted much.

Due to the nature of the decoding tree, an attacker could not throw off the synchronization. If an attacker tries broadcasting pulses that intentionally drift away from the sender's clock, the receiver will simply follow both paths, and successfully decode both the sender's and attacker's messages.

9. Concurrent Codes

The new results described in the previous sections suggests that it would be useful to consider a whole new branch of coding theory, which we will refer to as *concurrent codes*. Currently, there are two main types of codes: *error detecting codes* and *error correcting codes*. Concurrent codes will constitute a third branch of coding theory.

An error detecting code translates a message into a binary codeword. Then, if only a few bits are changed en route, the receiver will be able to detect that fact.

An error correcting code translates a message into a binary codeword. Then, if only a few bits are changed en route, the receiver will not only detect that fact, but will also be able to tell which bits were flipped, and so recover the original message intact.

A concurrent code translates each message into a binary codeword. Then, if several codewords are combined with a bitwise OR, the receiver can analyze the resulting string and recover all of the original codewords (and messages). In addition, if noise flips several bits from 0 to 1 (but never 1 to 0), the correct set of messages will still be recovered. The BBC algorithm is one example of a concurrent code, if there is a string of bits that are all 0 to start, a *location* is the position of a single bit in the string, and an *indelible mark* is a change of a bit from 0 to 1. This code, for any particular hash function, message length, k, and codeword length, can be called a *BBC code*.

Concurrent codes deal with a problem that is in some ways easier and in some ways harder than traditional codes. The new problem is easier in the sense that all noise is assumed to be one way. The noise (or an attacker) can change 0 bits to 1, but not vice versa. This is a realistic model, as discussed previously.

In another sense, this new problem is harder. Traditional error correcting codes can never act as concurrent codes. If two codewords X and Y from an error correcting code are combined with a bitwise OR, then at best the error correcting code will interpret the result as either the codeword X corrupted with many bit flips, or as the codeword Y corrupted with many bit flips. Such algorithms are inherently unable to interpret the result as containing both X and Y. In this sense concurrent coding theory deals with a more difficult problem than traditionally addressed.

10. Analysis of Random Codebooks

In developing a theory of concurrent codes, it is useful to start by analyzing the performance of an idealized *random codebook*, under an assumption of the availability of unbounded memory and computational power for encoding and decoding.

When several codewords are combined with a bitwise OR, the resulting bit string is called a *packet*. Assume the packet is decoded in the simplest, brute-force way: it is bitwise ORed with each codeword in the codebook. If the OR of the packet with a codeword equals that packet, then we say that codeword is contained in that packet.

If any set of messages is encoded into a single packet, and then the packet is decoded, clearly all of the original messages will be recovered. However, some additional messages may also be recovered. These will be codewords whose 1 bits just happen to lie in the same locations as those in the original set. These additional decryptions are the *hallucinations*. The most important analysis of a given codebook would be the expected number of hallucinations for a set of randomly-chosen messages.

Suppose a codebook is created for m-bit messages by associating with each message a random codeword. A codeword will be generated by randomly and independently choosing each bit to be 1 with probability μ_c. This procedure could theoretically produce the same codeword for two different messages, but that would simply introduce additional hallucinations, which will be reflected in the calculations below. The variables are all listed and described in Appendix A. For the random codebook, the performance is a function of the message length m, the expansion e, and the desired limit for the expected number of hallucinations h. Here, m is the number of bits in the message before being encoded, which is assumed to be in the hundreds or thousands. The expansion e is the ratio of the number of bits in the codeword to the number of bits in the message, and could be a hundred, a thousand, or a million. The hallucination rate h will typically be very small, such as 2^{-60}, which says that the probability of even a single hallucination arising from a given packet is astronomically low, so it is unlikely the system would give even a single hallucination in a lifetime of use (assuming the messages are generated randomly). Given these parameters (and assuming m and e are in the hundreds or larger), it is possible to calculate the performance of the system. We can calculate M_S, which is the maximum number of messages that can be sent simultaneously before the target hallucination limit is exceeded. We can calculate the bitrate ratio R, which is the maximum number of bits per second that the system can send, divided by the maximum bits per second for ordinary binary messages that are sent sequentially rather than simultaneously. So an R of 0.5 would imply that the penalty for allowing messages to be sent simultaneously is that bits can only be sent half as fast as they would otherwise be sent. It is also possible to calculate how dense the 1 bits should be in the codewords for optimum performance. We define k to be the expected number of codeword bits that are 1 in excess of m. In other words, each codeword contains on average $m + k$ bits that are 1, and the rest are 0. The following theorem gives the performance analysis for a random codebook.

Theorem 10.1. *For a random codebook, for any given choice of m, e, and h, where m and e are large (e.g. in the hundreds or larger), the average performance of the system for randomly-chosen messages sent simultaneously will be described by M_S, R, and k, which will be:*

$$M_S = \frac{-1}{\lg(1 - \frac{m - \lg(h)}{me})} \tag{10.1}$$

$$R = \frac{M_S}{e} \tag{10.2}$$

$$k = em(1 - (1 - \mu_p)^{\frac{1}{M_S}} - m \tag{10.3}$$

Proof. If the mark density for a codeword is μ_c, and if a packet consists of M_S randomly-chosen codewords combined with a bitwise OR, then the mark density for a packet can be found as follows. A given codeword bit is 1 with probability μ_c, so it is 0 with probability $1-\mu_c$. The corresponding bit in the packet will be 0 only if all M_S codewords had a 0 in that position, so the packet bit is 0 with probability $(1-\mu_c)^{M_S}$. Therefore, the packet bit will be 1 with probability

$$\mu_p = 1 - (1 - \mu_c)^{M_S} \tag{10.4}$$

Solving that equation for μ_c gives

$$\mu_c = 1 - (1 - \mu_p)^{\frac{1}{M_S}} \tag{10.5}$$

Multiplying this by the codeword length em gives the expected number of 1 bits in a codeword:

$$em(1 - (1 - \mu_p)^{\frac{1}{M_S}} \tag{10.6}$$

subtracting m gives k, the expected number of 1 bits in each codeword in addition to the message length m:

$$emk = em(1 - (1 - \mu_p)^{\frac{1}{M_S}} - m \tag{10.7}$$

If a codeword is chosen at random, it will be a hallucination if all of its 1 bits are in the same locations as 1 bits in the packet. So the probability of a 1 bit in a packet, μ_p, raised to the power of the number of 1 bits in a codeword will give the probability that a particular codeword is a hallucination:

$$(\mu_p)^{em(1-(1-\mu_p)^{\frac{1}{M_S}})} \tag{10.8}$$

Although the exponent in that expression is an expected value rather than an exact value, its variance will be low (because there are few ones in a codeword), and so using the expected value in the exponent will give a close approximation to the probability that a codeword is a hallucination.

If that is the probability of a single, random codeword being a hallucination, then the expected number of hallucinations from the packet, h, is simply that times the total number of codewords, 2^m.

$$h = 2^m(\mu_p)^{em(1-(1-\mu_p)^{\frac{1}{M_S}})} \tag{10.9}$$

Actually, the multiplier should have been $2^m - M_S$ rather than 2^m, because a message that was intentionally chosen can't be a hallucination. But when both m and M_S are in the hundreds or thousands, $2^m - M_S$ is extremely close to 2^m, and so multiplying by the latter gives almost exactly the correct answer. Solving this equation for M_S yields the maximum number of messages that can be sent simultaneously before the hallucination rate exceeds the desired bound:

$$M_S = \frac{\lg(1-\mu_p)}{\lg\left(1-\frac{\lg(h)-m}{em\lg(\mu_p)}\right)} \tag{10.10}$$

where lg() is the logarithm base 2. It is useful to consider the maximum achievable bitrate of sending the bitwise OR of concurrent code codewords like this, compared to the bitrate if the messages had simply been sent in binary one after the other. Let the bitrate efficiency ratio R be the ratio of these bitrates. In other words, if $R = 1/10$, that means that the penalty for allowing messages to be sent simultaneously is a 90% reduction in the amount of information that can be sent. Recall that an m-bit message is encoded as a codeword of length em, so a simple binary transmission of raw messages during that period would transfer at most em bits of information. On the other hand, if a concurrent code is used, at most M_S messages could be sent simultaneously before the hallucination rate exceeds the target, and so at most mM_S bits could be transferred. Therefore, the bitrate efficiency ratio is $R = (mM_S)/(em) = M_S/e$. Substituting in equation 10.10 gives:

$$R = \frac{M_S}{e} = \frac{\lg(1-\mu_p)}{e\lg\left(1-\frac{\lg(h)-m}{em\lg(\mu_p)}\right)} \tag{10.11}$$

Equation 10.11 is important. Given a packet bit density μ_p, a message length m, a message expansion e, and a desired hallucination rate h, this equation gives what fraction of the available bitrate can be achieved before exceeding the desired hallucination rate. For maximum efficiency, the packet density μ_p should be chosen to maximize R. This optimization must be done numerically for small values of e, but in the limit for large e the equation is greatly simplified:

$$\lim_{e\to\infty} R = \frac{-m\lg(\mu_p)\lg(1-\mu_p)}{\lg(h2^{-m})} \tag{10.12}$$

Note that μ_p only appears twice, and that R is maximized when $\lg(\mu_p)\lg(1-\mu_p)$ is maximized, which occurs when $\mu_p = 1/2$. Furthermore, even for fairly small values of e such as $e = 100$ (where $m = 200$ and $h = 2^{-60}$) the optimal value of e is already equal to 0.5 to one decimal place of precision, and it converges to exactly 0.5 in the limit as e grows large. Therefore, in almost all cases it can be assumed that $\mu_p = 1/2$. Under that assumption, equation 10.10 reduces to:

$$M_S = \frac{-1}{\lg(1-\frac{m-\lg(h)}{me})} \tag{10.13}$$

and equation 10.11 reduces to:

$$R = \frac{M_S}{e} = \frac{-1}{e\lg(1-\frac{m-\lg(h)}{me})} \tag{10.14}$$

This completes the derivation of all three equations that were to be proved. □

Using these equations it is possible to predict how efficient the random codebook can be. Figure 5 gives the results for both a random codebook and a BBC codebook (which is analyzed in the next section). In this table, the first three columns give various choices for the parameters, and the next 6 columns give the performance that results from those choices. The message is m bits long, the codeword is e

times as long as the message, and the goal is to limit the expected number of hallucinations to h. The gray cells show parameter choices that differ from the example case of $m = 1000$, $e = 100$, $\lg(h) = -60$. This gives an idea of how the performance is affected by each of the three parameters.

The results for the random codebook are surprisingly positive. In one case, the sender can send over 65,000 encoded messages simultaneously, combining them with a bitwise OR, and the receiver will recover every single one of them, with practically no hallucinations. The expected number of hallucinations is 2^{-60}, which is approximately 10^{-20}, which means that not even a single hallucination is likely to be seen in an entire lifetime.

Furthermore, this amazing ability to understand simultaneous messages comes at very little cost. The receiver wastes less than half of the available throughput by using this coding system instead of just sending unencoded messages sequentially. In most cases, the bitrate efficiency R is around 0.65, which means that a random codebook can send 65% as many bits per second as simple, unencoded binary communication, while still retaining an extremely low hallucination rate, and allowing many messages to be sent simultaneously. As mentioned before, concurrent codes are most likely to be used in hybrid systems, where this code is only used once to send a random session key, and all communication thereafter uses traditional, shared-secret, spread spectrum. In that case, a slight inefficiency in the first hundred bytes is very acceptable. Although BBC will be analyzed in the next section, note that its efficiency is around 40%, which is also very acceptable.

It is also interesting that the channel is maximally utilized when the output looks very much like ordinary messages. When compressed messages are sent sequentially, the stream of bits contains an equal number of 1s and 0s in a random-looking order. When a random codebook is used with enough simultaneous messages to achieve the optimal $\mu_p = 1/2$, the stream of bits contains an equal number of 1s and 0s in a random-looking order. It might be thought that such a high density of 1s would prevent error-free decoding, but surprisingly, that is not the case.

The first two lines of the table are just there to illustrate extreme cases, and to give good parameters for demonstrations of the system. They would be totally insecure in practice. Since the message length is only $m = 8$, an attacker can simply broadcast all 256 possible messages to ensure the receiver gains zero information.

All of these results can be summarized in a simple statement that holds true for all but the most extreme parameter choices. The final conclusion for random codebooks is surprisingly simple:

If you choose

- m = message length
- c = codeword length
- k = $\log_2$(maximum expected number of hallucinations allowed)

Then for optimal performance

- Generate each codeword with an average of $k + m$ bits set to 1
- The hallucination rate goal will be met as long as the number of messages sent simultaneously, M_S, is such that at most half the bits in the packet are 1
- This bound is met when:

$$M_S = \frac{-1}{\lg(1 + \frac{k-m}{c})} \tag{10.15}$$

Of course, all of these results are for an idealized, randomly-generated codebook. That is impossible to do in practice, since even for messages with a mere $m = 200$ bits, there would be 2^{200} codewords to generate and to check during decoding, which is impractical. Practical, efficient concurrent codes can be generated using Universal Concurrent Codes (described in a later section), or by using the BBC codes, which are analyzed in the next section.

11. Analysis of BBC Codes

Recall that the BBC algorithm works by making *indelible marks* at *locations* chosen by hashes of prefixes of a padded message (padded with some number of zero bits at the end). Abstracting away from the radio frequency details, we can define a *BBC code* to be a set of codewords, where the codeword corresponding to a message is a binary string of mostly 0 bits, with 1 bits in the positions chosen by a hash of each prefix of the padded message. In other words, for messages of length m, codewords of length em, and k padding bits, the codeword C corresponding to the message M has its ith bit defined as:

$$C_i = \begin{cases} 1 & \text{if } i = H(M'_{1..j}) \text{ for some } j \\ 0 & \text{otherwise} \end{cases} \tag{11.1}$$

where M' is the message M with k zero bits appended to the end.

For random codebooks, it was assumed that the receiver had infinite computational power, so the only concern was h, the expected number of hallucinations after all the decoding had been done. In contrast, BBC is designed to be implementable in practice, so it is necessary to be concerned not only with h, but with w, the average number of hallucinations at any given point in the decoding tree. The expected computational cost of decoding for large messages is then $\theta(wmM_S)$.

This is illustrated in figure 6, which is an example BBC decoding tree. In this example, there are 3 intentional messages (black circles) and a number of partial hallucinations (white circles), which are all eliminated by the end (far right edge). In this case, the code successfully eliminated hallucinations at the end, but the receiver still had to perform calculations for every partial hallucination. In this diagram, for any given column in the center region of the tree, there are an average of 6 partial

Parameter Choices			Random Codebook			BBC Codebook		
m	e	lg(h)	M_S	R	k	M_S	R	k
8	23	-24	3	0.1577	24	3	0.1339	16
8	15	-24	2	0.1490	23	2	0.1358	15
100	100	-60	42	0.4297	60	28	0.2877	40
1000	100	-60	65	0.6504	60	38	0.3894	41
10000	100	-60	68	0.6855	59	40	0.4038	41
1000	5	-60	2	0.5818	59	1	0.3904	38
1000	10	-60	6	0.6186	59	3	0.3902	39
1000	100	-60	65	0.6504	60	38	0.3894	41
1000	1000	-60	653	0.6536	59	388	0.3886	43
1000	10000	-60	6538	0.6539	60	3878	0.3879	45
1000	100000	-60	65390	0.6539	59	38708	0.3871	47
1000	100	-120	61	0.6154	119	37	0.3758	79
1000	100	-60	65	0.6504	60	38	0.3894	41
1000	100	-30	66	0.6695	29	39	0.3966	22
1000	100	-20	67	0.6761	19	39	0.3991	15
1000	100	-10	68	0.6828	9	40	0.4016	9

FIGURE 5. For particular choices of m, e, and h, this table gives the performance of random codebooks and BBC codebooks, as measured by the number of messages that can be sent simultaneously, M_S, the bitrate efficiency, R, and the number of 1 bits per codeword in excess of the message size, k. Choices are gray when they differ from the example of $m = 1000, e = 100, \lg(h) = -60$.

hallucinations and 3 messages. So in an average column, there are 9 nodes, 3 of which are the actual message, so the amount of calculation required to consider each node is $9/3 = 3$ times the computational cost as if only messages were considered. This implies a *workload factor* of $w = 3$, meaning there are $w = 3$ strings to consider (both messages and partial hallucinations) per intentional message, on average, for most of the tree.

Note that the tree has 3 regions. In the center region, each column has an average of wM_S nodes. At the far left, there is a small region where each column has fewer than that many nodes. This is because some messages may have the same first few bits, and the number of hallucinations takes some time before its exponential growth reaches its steady state value. At the far right, past the dotted line, the algorithm switches from decoding the message itself to checking the zeros that were appended to the message. From that point on, the number of hallucinations cannot grow, and will on average shrink exponentially, with any given hallucination dying out with probability $1 - \mu_p$ at each step. Since the left and right regions will typically be small compared to the middle region, the computational cost for decoding will be $\theta(mwM_S)$.

It is easy to calculate the steady state workload factor w, from the fact that in the steady state, the number of partial hallucinations (white nodes in one column) should on average remain unchanged in the next column. If a particular column has on average wM_S nodes, of which M_S are message prefixes and $(w-1)M_S$ are

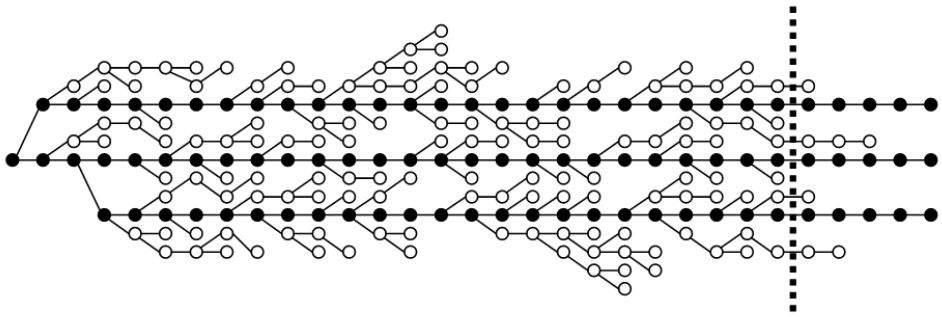

FIGURE 6. An example decode tree for BBC showing true messages (black circles) and hallucinations (white). Although no hallucinations survive until the end, there are partial hallucinations that require the receiver to expend extra computation. In this example, $w = 3$, because an average of 1/3 of the nodes in each column are black, so the receiver must expend 3 times the computation as would be necessary if there were no partial hallucinations.

partial hallucinations, then the next column to the right will have those same M_S messages plus several hallucinations. Each hallucination node at one time step gives rise to an average of $2\mu_p$ nodes on the next step, because there is a probability μ_p that appending a 0 will hash to a location containing a mark, and a probability μ_p that appending a 1 will hash to a mark location. Each message gives rise to a hallucination with probability of only μ_p, since the other possible bit to append gives a legitimate message prefix. If this is at steady state, then w should remain unchanged on the next time step, so we can equate the number of hallucinations on one time step to that on the next step:

$$(w-1)M_S = M_S\mu_p + 2(w-1)M_S\mu_p \tag{11.2}$$

Solving that for w gives:

$$w = \frac{1-\mu_p}{1-2\mu_p} \tag{11.3}$$

Note two important properties of this equation. First, when $\mu_p = 1/2$, the value of w goes to infinity. This means that the number of partial hallucinations will grow exponentially and never reach a steady-state limit, whenever $\mu_p \geq 1/2$. Second, when $\mu_p = 1/3$, the workload factor is a very reasonable $w = 2$. This means that the receiver will have to process an equal number of message nodes and hallucination nodes, and so the computational requirements are merely double the case of no hallucinations at all.

In the case of a random codebook, the optimal number of final hallucinations was reached at exactly $\mu_p = 1/2$. In a BBC codebook, "optimal" performance is achieved at a μ_p slightly below 1/2. The difference is because the computational cost is taken into account (as well as the algorithm being inherently less efficient). As μ_p grows from 1/3 to 1/2, the workload factor grows from a small $w = 2$ to $w = \infty$. There is therefore a tradeoff: the sender can manage to send more bits

in exchange for more computation on the receiver's side. But increasing μ_p from 1/3 to 1/2 gives only a 50% increase in bitrate, with an enormous increase in computational cost that is exponential in the size of the message. Similarly, a 25% bitrate increase from $\mu_p = 1/3$ to $\mu_p = 1.25/3$ increases the workload to $w = 3.5$, and a 40% bitrate increase from $\mu_p = 1/3$ to $\mu_p = 1.4/3$ increases the workload to $w = 8$. The user will have to determine how important bitrate is compared to the cost of computation. But typically, BBC will only be used to send one or two hundred bytes (e.g. a signed, random, session key), before communication switches to traditional, shared-secret spread spectrum. Therefore, we recommend using a target of $\mu_p = 1/3$ for BBC codes, which comes close to maximizing bitrate, while keeping computational costs at a very reasonable $w = 2$. All further analysis here will make that assumption. Therefore, the performance of BBC can summarized by the following theorem.

Theorem 11.1. *For a BBC codebook, choosing $w = 2$, and for any given choice of m, e, and h, where m and e are large (e.g. in the hundreds or larger), the average performance of the system for randomly-chosen messages sent simultaneously will be described by M_S, R, and k, which will be:*

$$M_S = \frac{\ln(2/3)\ln(3)}{\ln\left(1-\frac{1}{em}\right)W\left(\frac{3^m \ln(2/3)\ln(3)}{h\ln(1-\frac{1}{em})}\right)} \tag{11.4}$$

$$R = \frac{M_S}{e} \tag{11.5}$$

$$k = \log_3\left(\frac{\ln(2/3)\ln(3)}{h\ln\left(1-\frac{1}{em}\right)W\left(\frac{3^m \ln(2/3)\ln(3)}{h\ln(1-\frac{1}{em})}\right)}\right) \tag{11.6}$$

where the function W is the product-log function. `ProductLog[y]`.

Proof. The final hallucination rate, h, is now easy to calculate. At the dotted line, the column where decoding finishes with the message and starts processing the appended zeros, there will be an expected $(w-1)M_S$ hallucinations. After that point, no new hallucinations can be generated, and each of the existing ones will survive each additional time step with probability μ_p. Therefore the expected number of hallucinations remaining after all k zero bits have been processed is:

$$h = (w-1)M_S(\mu_p)^k \tag{11.7}$$

Solving that for k and substituting our choice of $\mu_p = 1/3$ and $w = 2$ gives:

$$k = \log_3\left(\frac{M_S}{h}\right) \tag{11.8}$$

Given this value for k, it is now possible to calculate the number of messages M_S that can be sent simultaneously to obtain the chosen $\mu_p = 1/3$. Each codeword is generated by starting with an em-bit codeword of all zeros, and then $m + k$ times setting a randomly-chosen bit to 1. So if M_S messages are sent simultaneously, the packet can be thought of as being generated by starting with an em-bit packet of all zeros, and then $(m+k)M_S$ times setting a randomly-chosen bit to 1. During a single one of those rounds, a given bit will be chosen to become 1 with probability $1/(em)$, and so will remain unchanged with probability $1 - 1/(em)$. So after all $(m+k)M_S$

rounds, a given bit will still be zero at the end with probability $(1-1/(em))^{(m+k)M_S}$. Therefore the final density of 1 bits will be 1 minus that value:

$$\mu_p = 1 - \left(1 - \frac{1}{em}\right)^{(m+k)M_S} \tag{11.9}$$

Substituting in the expression for k in equation 11.8 and the choice of $\mu_p = 1/3$ and $w = 2$ and solving the resulting equation for M_S yields:

$$M_S = \frac{\ln(2/3)\ln(3)}{\ln\left(1 - \frac{1}{em}\right) W\left(\frac{3^m \ln(2/3)\ln(3)}{h \ln(1-\frac{1}{em})}\right)} \tag{11.10}$$

where the function $W(y)$ is the product-log function, which is defined as the result of solving the equation $y = xe^x$ for x in terms of y, and is implemented in Mathematica by the function `ProductLog[x]`. Note that the logs are now natural logarithms because of the definition of W. Then, as before, $R = M_S/e$:

$$R = \frac{\ln(2/3)\ln(3)}{e \ln\left(1 - \frac{1}{em}\right) W\left(\frac{3^m \ln(2/3)\ln(3)}{h \ln(1-\frac{1}{em})}\right)} \tag{11.11}$$

Finally, substituting equation 11.10 into equation 11.8 gives the final result for k:

$$k = \log_3\left(\frac{\ln(2/3)\ln(3)}{h \ln\left(1 - \frac{1}{em}\right) W\left(\frac{3^m \ln(2/3)\ln(3)}{h \ln(1-\frac{1}{em})}\right)}\right) \tag{11.12}$$

This is the final equation that was to be proved. □

Equations 11.10, 11.11, and 11.12 give the expected performance for BBC, and were used to generate the table in figure 5. Note that the performance of BBC is comparable to that of an ideal, random codebook in most cases. Whereas a random codebook achieves a bitrate efficiency of about 65% in most cases, a BBC codebook achieves about 40%. It is acceptable to send data at half the rate of non-jam-resistance communication, especially if this halving of the bit rate only occurs while sending a session key, and for the rest of the session all communication will be at the full rate.

12. The Universal Concurrent Code Algorithm

This section presents another algorithm for the concurrent codes, the *universal concurrent code algorithm*. Several theorems are proved for this, including its universality, and its relationship to monotone Boolean functions and to monotone Boolean circuits. It is useful to start with basic definitions.

Definition 12.1. a *monotone Boolean function* (MBF) is any function $f : \{0,1\}^n \to \{0,1\}$ such that:

$$\forall x, y \in \{0,1\}^n : f(x) \le f(x \vee y) \tag{12.1}$$

An MBF is a function where increasing input bits from 0 to 1 will either leave the output unchanged or will increase it from 0 to 1. A great deal of research has been done on such functions [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96]. For every MBF

that can be evaluated in polynomial time, we define a corresponding concurrent code:

Definition 12.2. The *concurrent codes* are defined to be those codes that can be generated in the following manner. For any monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ that can be evaluated in deterministic polynomial time, the corresponding concurrent code is:

$$C = \{x \in \{0,1\}^n \mid f(x) = 1 \wedge [\forall y \in \{0,1\}^n : (x \wedge y \neq x) \Rightarrow f(x \wedge y) = 0] \quad (12.2)$$

Under this definition, a concurrent code is a set of codewords, where each codeword x has the property that $f(x) = 1$ and that changing any of the 1 bits to 0 in x will change this to $f(x) = 0$. So the codewords are the local minima of the support of f.

It is also useful to define what we mean by a probabilistic algorithm decoding efficiently.

Definition 12.3. we define a code C to be *efficiently decodable* if there is a probabilistic algorithm that, given a packet which is an arbitrary bit vector, is guaranteed to return in polynomial time a codeword from C that is contained in the packet (if any exist), where the codeword is chosen according to a probability distribution that is nonzero for every codeword in the packet and zero for all other bit vectors. If no codewords are contained in the packet, it must discover that fact in polynomial time.

Note that this definition means the decoder must never make an error, is guaranteed to find at least one codeword quickly, and has at least some probability of finding any of the other codewords present. However, it does allow a non-uniform probability distribution that could give some codewords an exponentially-small probability of being chosen. Ideally, a concurrent code would be chosen that avoids this problem. This is discussed in more detail below.

The most important theoretical result in this paper is the following theorem.

Theorem 12.4. *The family of concurrent codes is exactly those codes that can be efficiently decoded.*

Proof. To prove this, it will first be shown that every concurrent code can be efficiently decoded, and then that every efficiently-decodable code is a concurrent code.

To show that every concurrent code can be efficiently decoded, first note that the *universal concurrent code algorithm* given in algorithm 3 runs in linear time, plus n calls to the f function, because of the structure of the FOR loop. Therefore, if f runs in polynomial time, so does the universal algorithm.

One invariant is that at the end of each time through the FOR loop, $f(M) = 1$. This is because it tries to set one bit of M to 0, but if that single change would cause $f(M) = 0$, then it reverts that change. Therefore, the M that is eventually returned is such that $f(M) = 1$.

Furthermore, the M that is returned has the property that if even one of its bits is changed from 1 to 0, that would change $f(M)$ from 1 to 0. This property is because the algorithm is greedy, setting every bit to 0 that it possibly can. Thus,

by definition, the M that is returned is codeword in the code defined by the given f function.

It is also true that every codeword contained in the packet has a nonzero probability of being returned. If a codeword W is contained in the packet, consider the case where the sequence L consists first of the positions of all the 0 bits in W (in any order), followed by the positions of all the 1 bits in W (in any order). As the FOR loop works through all the 0 bit positions, setting those bits to 0 will still allow $f(M) = 1$, because W will still be contained in M after those bits are zeroed. Therefore the algorithm will zero out those bits. Then, as it goes through the positions of the 1 bits, it will be forced to leave them as 1 bits, because setting even a single one of them to 0 would eliminate W from the packet, and would only leave codewords in the packets that are subsets of W. But such subsets must not exist, otherwise W would not be a codeword. Therefore, the algorithm is guaranteed to return W. Since there is a nonzero probability of such an L being chosen, there is a nonzero probability for the return of every codeword in the packet.

The M that is returned will be a codeword in the codebook defined by the function f, will be returned in polynomial time, and will ensure that every codeword in the packet has nonzero probability. Thus, by employing the universal algorithm, every concurrent code is efficiently decodable.

Now it must be shown that every efficiently-decodable code is a concurrent code. This is proved by constructing the f that defines the concurrent code. Consider an arbitrary code C that has some method for efficiently decoding it. By the definition of efficient decoding, it must be possible to decide in polynomial time whether an arbitrary packet contains any codewords. Define $f(M)$ to be 0 if there are no codewords in M, and 1 otherwise. This f can be evaluated in polynomial time, and the concurrent code generated by it will be C. Therefore, every efficiently-decodable code is a concurrent code. □

It is interesting that there is a bijection between the set of polynomial-time monotone Boolean functions and the set of concurrent codes. So without loss of generality, the search for useful concurrent codes can be limited to simply searching for good MBFs. There is another well-known equivalence that may be relevant. The following two theorems are known from numerous publications, and are given here for completeness.

Definition 12.5. a *monotone Boolean circuit* (MBC) is a feedfoward Boolean circuit consisting only of AND and OR gates.

Theorem 12.6. *Every MBC implements an MBF, and every MBF can be represented by an MBC.*

Theorem 12.7. *There exist polynomial-time MBFs for which the smallest MBC is exponentially large.*

The fact that MBFs and MBCs exhibit equal computational power suggests that the search for good concurrent codes should include the consideration of Boolean circuits containing only AND and OR gates. Every such circuit (if of polynomial size) will define a concurrent code, and so these can be explored. And many MBFs do have efficient circuit implementations, so this may be a fruitful area of research.

Unfortunately, not all polynomial-time MBFs have polynomial-sized MBCs. Therefore, the search for good concurrent codes should not be limited to such circuits.

Algorithm 3 UniversalDecode(P,f)

This is the Universal Concurrent Code algorithm. This probabilistic function returns one codeword from the n-bit packet P. The returned codeword will be an element of the codebook defined by the polynomial-time monotonic Boolean function f. This runs in linear time (plus the time for n calls to f). Every codeword in the packet has a nonzero probability of being returned, and all other bit vectors have zero probability.

if $f(P) = 0$ **then**
  return "The packet contains no codewords"
end if
$M \leftarrow P$
$L \leftarrow$ the list $1, 2, ..., n$ in a randomly-permuted order
for $i \leftarrow 1 \ldots n$ **do**
  $M[L[i]] \leftarrow 0$
  if $f(M) = 0$ **then**
    $M[L[i]] \leftarrow 1$
  end if
end for
return M

Other monotone functions should be considered as well. We are currently researching a wide range of MBFs (both MBCs and others) to find concurrent codes that give good performance, low hallucination rates, and efficient decoding (with almost uniform probabilities) when decoded with the universal algorithm.

It is desirable that when the universal algorithm decodes a packet, each of the codewords present would have roughly equal probabilities of being returned. That allows decoding to be done quickly by running the algorithm repeatedly. Even if there are rare packets for which this property fails, it would be useful if it were difficult for an attacker to greatly decrease the probability of a legitimate message. To see these issues more clearly, it is useful to have an example of a packet where one message has much lower probability than all the others. The following is such an example (with zeros represented as dashes for clarity):

```
Codeword 1:   111111------
Codeword 2:   -111111-----
Codeword 3:   1-1111-1----
Codeword 4:   11-111--1---
Codeword 5:   111-11---1--
Codeword 6:   1111-1----1-
Codeword 7:   11111------1
Packet:       111111111111
```

In this example, by symmetry, messages 2 through 7 are all equally likely to be found. Codeword 1 is much less likely. Note that if the first bit position chosen happens to lie on the left half of the packet, then it will zero out that bit, because one of the messages 2 through 7 will have a 0 there, and so $f(M)$ will continue to equal 1 even after that bit is zeroed out. Once that happens, there is no way that codeword 1 can be returned at the end. The only way to get codeword 1 to be

returned is if the sequence always chooses position $x+6$ before choosing position x, for all $x \in \{1, 2, 3, 4, 5, 6\}$. The probability of this happening in a given run is only 2^{-6}. If many codewords were constructed in this way, the probability of codeword 1 being chosen would be exponentially small.

This example has several interesting properties. The codewords 2 through 7 all have a Hamming distance of 4 from each other, but a distance of only 2 from codeword 1. So they form a very small sphere around message 1, and are evenly distributed on the surface of that sphere. If the concurrent code happens to have its codewords spread out roughly evenly, then it will be unlikely that many codewords will exist at the centers of tightly-packed spheres of other codewords. If a few such smothered codewords do exist, and the code is used for jam-resistant communication, they may be sent rarely during normal communications. And if one is actually sent by a legitimate user, it may be difficult for an attacker in real time to quickly recognize that fact, identify the sphere that will block it, and generate all the codewords on that sphere as an attack. None of this is certain, but it is suggestive that it may be possible to find monotone Boolean functions that generate useful concurrent codes that are difficult to attack in a jamming situation. Research in this area is ongoing.

13. BBC for RFID Applications

Radio Frequency Identifier (RFID) chips are currently an area of great interest [97, 98, 99, 100]. An RFID chip allows a wireless scanner (at a distance of multiple meters) to query the chip, which responds wirelessly with some kind of identifier code. Although RFID systems exist that contain batteries and small computers, the systems that are currently generating great interest are completely unpowered. They consist of a chip and a thin antenna which are powered only by the RF query signal itself.

RFID chips are currently used for managing logistics in the retail community by placing one chip on each palette of merchandize. This allows warehouses and trucks to remotely query their contents and identify which palettes are present.

However, the vision for the future of RFID is at a much lower scale. Retailers would like to embed an RFID chip in the packaging of individual items. In this scenario, every can of soup or package of toothpaste in a store would contain an RFID chip, and a reader on the ceiling would be able to monitor the inventory in real time. In addition, every item in a grocery cart could be scanned simultaneously in a fraction of a second, greatly speeding checkout times. The RFID chip would act as a faster and more convenient alternative to the Universal Product Code (UPC) barcode that is currently printed on most items.

This vision of the future is currently limited by the cost of the chips, which is expected to drop rapidly as they become mass produced. Unfortunately, there is another limitation that appears more difficult to solve. If the system is scaled up from one chip per palette to one chip per item, the number of chips within the range of a ceiling-mounted reader will grow tremendously. There is now appearing to be a problem with the chips jamming each other. If the chips are powered by the signal from the sensor, then they must all respond simultaneously. If there are many of them, they will tend to jam one another. If the code is meant to be applied at the factory and uniquely identify every item produced, then the space of potential codes will be enormous. This suggests that the current binary codes in

use are not suited to this problem, and that traditional superimposed codes would not work well either. It appears that BBC codes and other concurrent codes could be the solution to this problem of accidental jamming.

14. BBC for Information Retrieval Applications

Superimposed codes are often used as part of various information retrieval search and query algorithms [101, 102]. They use the fact that it is easy to check whether or not a given codeword is contained in a given packet. However, BBC goes one step further, and allows all the codewords in a packet to be extracted efficiently. This suggests that BBC may be a useful new tool for increasing the speed of query processing. This section will show some ways that BBC can be applied for information retrieval.

Suppose that a database of documents, perhaps from the Web, is to be searched for pages related to terrorist car bombings. A typical query might be something like:

```
(attack OR bomb) AND car
```

This is a search for every document that contains the word "car" and also contains at least one of the words "attack" or "bomb". Suppose there are 2^n documents, each of which was assigned a unique identifier which is an n-bit string, and that identifiers were assigned to documents in a random order. What is the fastest way to perform this query?

A classical approach is to maintain a sorted list of the document IDs for every document containing the word "attack". A similar list could be stored for "bomb" and "car", with the ID length $n = \lceil lg(\text{number of documents}) \rceil = 5$.

```
documents containing "attack":  {00101, 01001, 11100}
documents containing "bomb":    {00101, 01011}
documents containing "car":     {00111, 01001, 11100, 11101}
```

A complete list of documents satisfying the query could be found by doing a merge of the "attack" and "bomb" lists (removing duplicates as they were merged), and then finding the intersection of that list with the "car" list. The simplest approach to finding the union or intersection of two lists would require time that is at least linear in the shorter of the two lists.

There are several problems here. Perhaps all three lists are extremely long, but the solution set for the entire query is very short. In that case, a great deal of computation must be performed per document in the solution set. Also, if the user wants only one document satisfying the query, it may require a large amount of computation before that one is found. The BBC code can help with these. Instead of storing a sorted list of document IDs, we will store a packet that contains the bitwise OR of the BBC encoding of those IDs:

```
Attack:  00011000011101101101000000011000010000011
Bomb:    00000100000001110000001100000000101000011
Car:     00101011010000111000111000011100011110011
```

These packets are slightly different from the most common way of forming signatures. It would be more typical to form a packet (signature) for each document, and have the packet contain a codeword for each word in the document. Here it is the other way around. There is a packet for each word, and it contains a codeword for every document that contains that word.

The Boolean operations of the query can now be performed directly on the packets themselves:

```
Attack:                    000110000111011011010000000110001000011
Bomb:                      000001000000011100000011000000010100011
Attack OR Bomb:            000111000111011111010011000110011100011
Car:                       001010110100001110001110000111001111001l
(Attack OR Bomb) AND Car: 000010000100001110000010000110001100011
```

As with a traditional superimposed code signature, or a traditional Bloom filter, the packet at the bottom is found by doing a bitwise OR of the "attack" and "bomb" packets, followed by a bitwise AND of the result with the "car" packet. However, unlike traditional signatures or Bloom filters, all of the document IDs contained in that bottom packet can be extracted very quickly. If a document satisfies the query, it is guaranteed to be found in the final packet. If it does not satisfy the query, the probability of being found in the final packet is very low, as controlled by k, the number of zeros appended to each ID, and by the density of the final packet.

Assume the final solution packet has already been created, and happens to have 1/3 or fewer of its bits set to 1. In that case a single solution document can be found in time proportional to the length of a single document ID (plus k), regardless of the length of the packet or the number of solution documents. A database of a billion documents could be addressed with 30-bit document IDs, so this decoding can happen very quickly, even if the solution packet is millions of bits long, and is densely packed with 1 bits (e.g. a density of 1/3). If the density were high (e.g. greater than 1/2), then the BBC search would fail. However, if the solution set is at most the same size as the set of documents containing one of the query terms, then the density should approximately equal the density of that query term's packet, and so would be sufficiently low.

Of course, the search speed wouldn't help if we had to physically construct the solution packet based on the query. But there is no need for that. Bits from that packet can be constructed on the fly as needed by the decoding algorithm. If the goal is to quickly find a single solution document, and the tree is only 30 layers deep, and the density is 1/3 (so $w = 2$), then only 60 bits of the packet need be calculated, even if the packet is millions of bits long. That requires looking at only 60 bits from each of the query term packets.

In other words, the solution packet (which is purely virtual) is decoded by BBC by looking at certain bits from the query term packets and performing the appropriate Boolean operations on them. Therefore, there is no need for all the packets to be the same length, nor for the same hash functions to be used in every packet. If a query term only exists in a small number of documents, then its packet can be made relatively short. A more common term will need a longer packet to store all of its documents IDs.

Note that the operations described so far are AND and OR operations. The NOT operation is somewhat different. A separate packet should be built for the NOT of every term. So the "NOT Attack" packet would contain the IDs of every document that *doesn't* contain the word "attack". The query would be preprocessed using de Morgan's law repeatedly to distribute each NOT over parenthetical expressions so the NOT operators only apply to query terms, rather than to entire subexpressions. Once this is done, searching proceeds as describe above.

There is a property of BBC search that is interesting. At each step of the search, then entire query is influencing the answer. In the simplistic search algorithm described first, the query (A OR B) AND C was performed by first performing (A OR B) and then performing the final AND. In a BBC search, the search can be thought of as discovering the first bit of the solution document ID, then the second bit, and so on, where the entire query is involved at every stage.

Another property of BBC search is that it allows very general queries. First, it does not require the query to be in a special form like conjunction normal form (CNF) or disjunction normal form (DNF). For example, BBC can directly handle a query such as:

```
A AND (B OR (C AND (D OR (E AND (F OR (...))))))
```

Such a query would grow exponentially large if converted into CNF or DNF. Second, BBC search can also apply in an even more general situation. For example, consider a long list of query terms, where the goal is to find any document containing the majority of those terms. Such queries cannot be represented efficiently as a Boolean expression. Even a simple query such as "contains a majority of A, B, C, D, E" expands into a very long Boolean query:

```
(A AND B AND C) OR (A AND B AND D) OR (A AND B AND E) OR
(B AND C AND D) OR (B AND C AND E) OR
(C AND D AND E)
```

The Boolean query is of size exponential in the number of search terms. However, there is a very simple Boolean circuit that, when given the inputs A, B, C, D, E outputs whether a majority of them are true. Such a circuit can be found by using a Boolean sorting circuit (as described in Knuth [103]), and then taking the middle bit of the output. Such a majority circuit can be built entirely of AND and OR gates without any NOT gates. So BBC can efficiently search for the solutions to this query. Every time BBC needs to look at one bit in the virtual solution packet, it will actually look at 5 bits (one each from the A, B, C, D, E packets), and pass them through the majority circuit. The output gives the corresponding bit in the virtual solution packet. The search is equally fast if the question is to find documents containing at least, say, 73% of the search terms instead of 50%. In this way, BBC search can support far more powerful queries than would be possible with Boolean expressions alone.

BBC search can be made more powerful in other ways as well. When doing a depth-first search of the BBC decoding tree, the algorithm could always take the 0 branch rather than 1, whenever it has a choice. If that path failed to give a solution, it would eventually backtrack and try the 1 path. In this case, the BBC decoder will output the IDs of every document in the solution set, and will output them in lexicographic order. This is interesting. It was suggested that document IDs be assigned to documents randomly. They could instead be assigned in some other order. For example, the documents with the best Google pagerank could be given the lowest ID numbers. In that case, BBC search would always return the best document of all those that satisfy the query. Even if there are millions or billions of satisfying documents, it would still return the best one in time linear in the document ID length.

Alternatively, the depth-first search could choose 0 and 1 choices randomly, only exploring the other path when it was forced to backtrack. In that case, the first document it finds would be a randomly-chosen member of the solution set. If the

user wants random samples of that set, this would be ideal. If the document IDs were assigned randomly, then the member of the solution would typically be chosen with close to uniform probability distribution.

Furthermore, BBC search can be extended with weightings and other fuzzy approaches. Suppose that the query is (A OR B), but the terms are weighted, and A has a greater weight than B. If BBC performs a complete depth-first search, it will eventually return every document containing either A or B. But whenever it must choose between two paths, if one path is supported only by the A packet and the other is supported only by the B packet, then it could always choose to follow the A-supported path first. In this way, it would be biased toward finding the solution documents with the highest fuzzy weighting first, and only find the low-weighted solutions later.

That form of weighted search was greedy: always pick the path with greatest fuzzy weight. More powerful searches could be performed instead, such as a uniform cost search of the BBC tree, or even an A* search [104] if fuzzy heuristics exist to guide the search. In every case, the goal would be to find elements of the solution set in an order such that the "better" documents are found before the "worse" documents. Naturally, complex fuzzy queries might be too difficult for a typical user to design by hand. But they could be automatically generated from other information, such as a list of documents that the user rated as relevant or not. That sort of automated query construction is beyond the scope of this paper. But it appears that once the fuzzy query has been generated, BBC could increase the speed at which it is processed.

Finally, it should be noted that when a large packet is stored on disk (or even in RAM, if the computer has limited cache memory), then the hash function can be optimized for cache, locality, and seek-time effects. It was assumed that in BBC, a hash of each prefix pseudorandomly chooses where in the packet a 1 bit will be placed. So, during the decoding of a single query solution, the algorithm will need to retrieve only a tiny fraction of the bits in each query term packet, but those bits will be spread out pseudorandomly and uniformly throughout the packet. For some disk systems, this could require that the entire packet be read into memory, even though only a tiny fraction is required.

There is an alternative. The packet could be partitioned into 2 sections, the first small and the second very large. For all message prefixes of length x or less, the hash function would choose a pseudorandom location in the first section to place the 1 bit. Then a hash of the entire first x bits would be used to choose a small block of contiguous bits from the second section of the packet. Finally, the remainder of the prefixes (each longer than x bits) would choose locations within that block pseudorandomly.

This is actually standard BBC, except with a modified hash function. If the blocks are chosen pseudorandomly, then a packet with many messages will typically choose many blocks, and so the 1 bits in that packet will be spread out uniformly, as is necessary for good performance. However, during the decoding of a single message, the algorithm would only need to fetch the entire first section of the packet (which is small) plus a single block from the second section (which is a small fraction of the second section of the packet). In this way, only a small fraction of the packet would need to be loaded from disk, and that small fraction would consist of

contiguous bytes, which would reduce seek time substantially (for a defragmented disk).

This modification to the hash function in BBC is very similar to the many partitioned and multi-level approaches used in existing signature algorithms [102].

15. Acknowledgements

The authors gratefully acknowledge the help of Martin Carlisle, Sean Butler, and Dino Schweitzer.

16. Appendix A: Notation

c	=	length of a codeword = length of a packet
d	=	expected number of marks per codeword
e	=	c/m=codeword expansion
g	=	processing gain = number of codewords to OR together to get $\mu_p = 1/2$
h	=	$M_R - M_S$=hallucination rate (expected number of hallucinations per packet)
k	=	in BBC, the number of 0 bits appended to the message
m	=	length of a message
M_S	=	number of messages intentionally put into a packet by senders
M_R	=	number of messages receiver gets from a packet (all)
M_H	=	$M_R - M_S$=number of hallucinations in a packet
M_A	=	number of complete messages intentionally sent by attacker
R	=	M_S/e=bitrate ratio (max message bits per second for sequential broadcast) / (max for simultaneous)
s	=	d/m=mark factor (expected number of marks per message bit)
w	=	workload factor = (expected number of partial hallucinations and messages at one point in decoding) / M_S
x	=	a binary string (used in algorithms and examples)
μ_c	=	d/c=mark density (probability a codeword bit is 1)
μ_p	=	mark density (probability a packet bit is 1)

17. Appendix B: Mathematica Code

```
(*
 This Mathematica file generates all the major equations and tables in this paper.
 Variables:
   m = number of bits per message
   e = (number of bits per codeword)/(number of bits per message)
   u = probability a packet bit is 1 (mup)
   M = number of codewords ORed to make a packet
   h = number of hallucinations from a packet expected
   w = workload factor = (number of messages + number
       of hallucinations during decoding)/(number of messages)
*)

Off[General::"spell1"];
Off[FindMaximum::"nnum"];
Off[Solve::"ifun"];
Off[InverseFunction::"ifun"];
output[codebook_, var_, equation_] :=
    (eq = Append[eq, {equation, var, codebook}]);
eq = {};
```

```
output["Rnd", "= Prob a given codeword bit is set",
        ans = (uc //. Solve[u == 1 - (1 - uc)^M, uc][[1]])];
output["Rnd", "= Expected # of codeword bits that are 1", ans = ans*m*e];
output["Rnd", "= Prob a given codeword is a hallucination",
        ans = u^ans]; (*raising to expected value only approximates e.v.*)
output["Rnd", "= Expected # of hallucinations",
        ans = ans*(2^m)]; (*should use (2^m)-M, but it's almost the same*)

Msol = Solve[ans == h, M][[1]] // FullSimplify;
M[ee_, mm_, uu_, hh_] := (M //. Msol //. {e -> ee, m -> mm, u -> uu, h -> hh});
Mstar[e_, m_, h_] :=
    FindMaximum[M[e, m, u, h], {u, .5}][[1]]; (*M for u that maximizes M*)
Rstar[e_, m_, h_] := Mstar[e, m, h]/e; (*the bitrate ratio R, for the optimal u*)
uStar[e_, m_, h_] := FindMaximum[M[100, 100, u, 2^(-60)],
                                 {u, .5}][[2, 1, 2]]; (*the u that maximizes M *)
kStar[e_, m_, h_] := e*m*(1 - (1 - uStar[e, m, h])^(1/Mstar[e, m, h]))
                     - m; (* # of 1 bits per codeword, for the optimal u*)

output["Rnd", "= M", M[e, m, u, h]];
output["Rnd", "= R", M[e, m, u, h]/e];
output["Rnd", "= M, for u=1/2", M[e, m, u, h] //.
    {u -> 1/2, Log[2] -> 1, Log[x_] -> Lg[x], Lg[2] -> 1, Lg[1/2] -> -1}];
output["Rnd", "= R, for u=1/2", M[e, m, u, h]/e //.
    {u -> 1/2, Log[2] -> 1, Log[x_] -> Lg[x], Lg[2] -> 1, Lg[1/2] -> -1}];
output["Rnd", "= R, as e->Infinity",
    Limit[M[e, m, u, h]/e, e -> Infinity] // FullSimplify];

h =.; w =.; Ms =.; k =.;
BBCwsol = Solve[(w - 1) * Ms == Ms * mup + 2 (w - 1) * Ms * mup, w][[1]];
BBCksol = Solve[h == (w - 1)*Ms*(1/3)^k, k][[1]];
BBCMsol = Solve[mup == 1 - (1 - 1/(e *m))^((m + k)*Ms) //.
                BBCksol //. {w -> 2, mup -> 1/3}, Ms] // FullSimplify;
BBCMstar[ee_, mm_, hh_] :=
    (Ms //. BBCMsol[[1]] //. {e -> ee, m -> mm, h -> hh});
BBCRstar[e_, m_, h_] := BBCMstar[e, m, h]/e;
BBCkstar[ee_, mm_, hh_] := (k //. BBCksol //. {Ms -> BBCMstar[ee, mm, hh]} //.
    {w -> 2, mup -> 1/3, e -> ee, m ->mm, h -> hh});

output["BBC", "= w", w //. BBCwsol];
output["BBC", "= k", k //. BBCksol];
output["BBC", "= M_S", BBCMstar[e, m, h]];
output["BBC", "= R", BBCRstar[e, m, h]];
output["BBC", "= k", BBCkstar[e, m, h]];

eq // TableForm

(*generate the LaTeX code for one row in the table giving all results.*)
(*The columns in the table are
     m
     e
     lg(h)
     MS for random codebook
     R for random codebook
     MS for random codebook
     R  for random codebook
  *)
ts[x_, w_, p_] := If[p == 0,
      ToString[PaddedForm[Floor[x], w]],
      ToString[PaddedForm[N[x], {w, p}]]];

resultsTableRow[{m_, e_, lgh_}] := Block[{},
      " "
        <> If[m == 1000, "    ", "\gry"] <> ts[m, 7, 0] <> " & "
        <> If[e == 100, "    ", "\gry"] <> ts[e, 6, 0] <> " & "
        <> If[lgh == -60, "    ", "\gry"] <> ts[lgh, 3, 0] <> " & "
        <> ts[Mstar[e, m, 2^lgh], 5, 0] <> " & "
        <> ts[Rstar[e, m, 2^lgh], 5, 4] <> " & "
        <> ts[kStar[e, m, 2^lgh], 7, 0] <> " & "
        <> ts[BBCMstar[e, m, 2^lgh], 5, 0] <> " & "
        <> ts[BBCRstar[e, m, 2^lgh], 5, 4] <> " & "
        <> ts[BBCkstar[e, m, 2^lgh], 7, 0] <> " \\\\ \\hline\n"
      ];
```

```
tableCases = {
      {8, 23, -24},
      {8, 15, -24},
      {100, 100, -60},
      {1000, 100, -60},
      {10000, 100, -60},
      {1000, 5, -60},
      {1000, 10, -60},
      {1000, 100, -60},
      {1000, 1000, -60},
      {1000, 10000, -60},
      {1000, 100000, -60},
      {1000, 100, -120},
      {1000, 100, -60},
      {1000, 100, -30},
      {1000, 100, -20},
      {1000, 100, -10}
      };
StringDrop[StringJoin @@ (resultsTableRow /@ tableCases), -6] <> "whline"
```

References

[1] A. Belouchrani and M. Amin. Jammer mitigation in spread spectrum communications using blind source separation. *Signal Processing*, 80(4):723–729, April 2000.

[2] L. B. Milstein. Interference rejection techniques in spread spectrum communications. *Proc. IEEE*, 76(6):657–671, June 1998.

[3] 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.*, pages 679–683, 1998.

[4] 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, July 1994.

[5] E. G. Kanterakis. A novel technique for narrowband/broadband interference excision in ds-ss communications. In *MILCOM '94*, volume 2, pages 628–632, 1994.

[6] L. Li and L. Milstein. Rejection of pulsed cw interference in pn spread-spectrum systems using complex adaptive filters. *IEEE Trans. Commun.*, COM-31:10–20, January 1983.

[7] L. B. Milstein. Interference suppression to aid acquisition in direct-sequence spread-spectrum communications. *IEEE Transactions on Communications*, 36(11):1200–1207, November 1988.

[8] H. V. Poor and L. A. Rusch. Narrowband interference suppression in spread spectrum cdma. *IEEE Personal Communication Magazine*, 1:14–27, August 1994.

[9] T. Ristaniemi, K. Raju, and J. Karhunen. Jammer mitigation in ds-cdma array system using independent component analysis. In *Proc. IEEE Int. Conf. on Communications*, New York, USA, April 2002. To appear.

[10] 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)*, pages 103–107, October 1992.

[11] C. Bergstrom and J. Chuprun. Optimal hybrid frequency hop communication system using nonlinear adaptive jammer countermeasures and active fading mitigation. In *IEEE Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration*, volume 6.

[12] 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*, June 2003.

[13] I. Bergel, E. Fishler, and H. Messer. 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*, pages 303–307, May 2002.

[14] R. Blazquez and A. P. Chandrakasan. Architectures for energy-aware impulse uwb communications. *ICASSP*, March 2005.

[15] G.T Ahlstrm and D Danev. A class of superimposed codes for cdma over fiber optic channels. Technical Report LiTH-ISY-R-2543.

[16] S. Mariac and V. Lau. Multirate fiber-optic cdma: System design and performance analysis. *Journal of Lightwave Technology*, 16(1), January 1998.

[17] Leslie Rusch and H. Vincent Poor. Narrowband interference suppression in cdma spread spectrum communications. *IEEE Transactions on Communications*, 42(2/3/4):1969–1979, April 1994.

[18] K. Raju, T. Ristaniemi, J. Karhunen, and E. Oja. Jammer suppression in ds-cdms arrays using independent component analysis. *IEEE Transactions on Wireless Communications*, 5(1), January 2006.

[19] Z. Ye, D. Lee, G. Saulnier, and M. Medley. Anti-jam, anti-multipath spread spectrum ofdm system. In *Conference Record of the Thirty-Second Asilomar Conference on Signals, Systems & Computers, 1998*, volume 2, pages 1793–1797, November 1998.

[20] G. Saulnier, M. Mettke, and M. Medley. Performance of an ofdm spread spectrum communications system using lapped transforms. In *Proceedings of the IEEE Military Communications Conference, 1997 (MILCOM 97)*, volume 2, pages 608–612, November 1997.

[21] R. Blazquez, P. Newaskar, and A. Chandrakasan. Coarse acquisition for ultra wideband digital receivers. In *Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing*, volume IV, pages 137–140, Hong Kong, China, April 2003.

[22] Kevin J. Negus, John Waters, Jean Tourrilhes, Chris Romans, Jim Lansford, and Stephen Hui. Homerf and swap: Wireless networking for the connected home. *ACM Mobile Computing and Communications Review*, 2(4):28–37, October 1998.

[23] P. Newaskar, R. Blazquez, and A. Chandrakasan. A/d precision requirements for an ultra-wideband radio receiver. In *SIPS 2002*, pages 270–275, San Diego, CA, October 2002.

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

[25] R. Jean-Marc Cramer, Robert A. Scholtz, and Moe Z. Win. Evaluation of an ultra-wide-band propagation channel. *IEEE Transactions on Antennas and Propagation*, 50(5):561–570, May 2002.

[26] M. Z. Win and R. A. Scholtz. On the robustness of ultra-wide bandwidth signals in dense multipath environments. *IEEE Communications Letters*, 2(2):51–53, February 1998.

[27] M. Z. Win and R. A. Scholtz. On the energy capture of ultrawide bandwidth signals in dense multipath environments. *IEEE Communications Letters*, 2(9):245–247, September 1998.

[28] Moe Z. Win and Robert A. Scholtz. Ultra-wide bandwidth time-hopping spread-spectrum impulse radio for wireless multiple-access communications. *IEEE Transactions on Communications*, 48(4):679–691, April 2000.

[29] http://www.globalgadgetuk.com/Personal.htm.

[30] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols, 7th International Workshop Proc., Lecture Notes in Computer Science*, 1999.

[31] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.

[32] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), November 1999.

[33] M. Burmester and Tri Van Le. Secure communications in ad hoc networks. In *Proceedings of the 2004 IEEE Workshop on Infromation Assurance and Security*, pages 234–241, May 2004.

[34] M. Burmester and Tri Van Le. Secure multipath communications in mobile ad hoc networks. In *International Conference on Information Technology: Coding and Computing (ITCC 2004)*, April 2004.

[35] Y. Desmedt, R. Safavi-Naini, H. Wang, C. Charnes, and J. Pieprzyk. Broadcast antijamming systems. In *ICON '99: Proceedings of the 7th IEEE International Conference on Networks*, pages 349–355, Washington, DC, USA, 1999. IEEE Computer Society.

[36] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, 1976.

[37] Martin E. Hellman. An overview of public key cryptography. *IEEE Communications Magazine, 50th Anniversary Commemorative Issue*, pages 42–49, May 2002.

[38] R. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, 1979.

[39] Whitfield Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, May 1988.

[40] M. Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology - Pre-Proceedings of Eurocrypt '94*, 1995.

[41] M. Burmester and Yvo Desmedt. A secure and scalable group key exchange system. *Information Processing Letters,*, 94(3):137–143, 2005.

[42] Y. Desmedt and M. Burmester. Towards practical proven secure authenticated key distribution. In *Proceedings 1st ACM Conference on Computer and Communication Security*, pages 228–231, 1993.

[43] D. R. Stinson, Tran van Trung, and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference*, 86:595–617, 2000.

[44] S. Yekhanin. New locally decodable codes and private information retrieval schemes. *Electronic Colloquium on Computational Complexity*, TR06:127, 2006.

[45] S. Lumetta and S. Mitra. X-codes: Theory and applications of unknowable inputs. Technical Report CRHC-03-08 (also UILU-ENG-03-2217), UIUC Center for Reliable and High-Performance Computing.

[46] Hans Georg Schaathun Grard D. Cohen. Asymptotic overview on separating codes. Technical Report 2003-248.

[47] M. Taube. Superimposed coding for data storage. Technical Report Rept. No. 15.

[48] F. Rogers. [a review of] studies in coordinate indexing [by mortimer taube]. *Bull Med Libr Assoc.*, 42(3):380–384, July 1954.

[49] W. Kautz and R. Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, pages 363–377, 1964.

[50] D. Danev. Some constructions of superimposed codes in euclidean spaces. *Discrete Applied Mathematics*, 128(1):85–101, May 2003.

[51] Peter-Olof Anderson. Superimposed codes and bn-sequences. Technical Report LiTH-ISY-I-1108.

[52] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[53] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John 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*, pages 181–192, New York, NY, USA, 2005. ACM Press.

[54] A. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *Theor. Comput. Sci.*, 302(1-3):337–364, 2003.

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

[56] 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*, page 367, 1993.

[57] Arkadii D'yachkov, Vladimir Lebedev, Paul Vilenkin, and Sergi Yekhanin. Cover-free families and superimposed codes: Constructions, bounds, and applications to cryptography and group testing. In *IEEE International Symposium on Information Theory*, 2001.

[58] D. R. Stinson and R. Wei. Generalized cover-free families. *Discrete Mathematics*, 279:463–477, 2004.

[59] R. Wei. On cover-free families. Technical report, Lakehead University, 2006.

[60] Arkadii D'yachkov, Anthony Macula, David Torney, Pavel Vilenkin, and Sergey Yekhanin. New results in the theory of superimposed codes. In *roceedings of International Conf. on Algebraic and Combinatorial Coding Theory (ACCT)*, pages 126–136, 2000.

[61] Sergy Yekhanin. Sufficient conditions of existence of fix-free codes. *IEEE International Symposium on Information Theory*, June 2001.

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

[63] 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*, 306:223–243, 2003.

[64] 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)*, pages 95–99, 2002.
[65] S. Yekhanin. Some new constructions of optimal superimposed designs. In *Proceedings of International Conf. on Algebraic and Combinatorial Coding Theory*, pages 232–235, 1998.
[66] A. de Bonis and Ugo Vaccaro. Efficient constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. In *ESA*, pages 335–347, 2002.
[67] Thomas Erickson and Laszlo Gyorfi. Superimposed codes in rn. Technical Report LiTH-ISY-I-0818.
[68] Tayuan Huang and Chih wen Weng. A note on decoding of superimposed codes. *Journal of Combinatorial Optimization*, 7:381–384, 2003.
[69] D. Awduche and A. Ganz. Mac protocol for wireless networks in tactical environments. In *IEEE Military Communications Conference, MILCOM-96*, October 1996.
[70] Lars-Inge Alfredsson. A class of superimposed codes for cdma over fiber optic channels. Technical Report LiTH-ISY-I-1045.
[71] Fidel Cacheda and Ángel Viña. Superimposing codes representing hierarchical information in web directories. In *WIDM*, pages 54–60, 2001.
[72] Y. Desmedt. A high availability internetwork capable of accomodating compromised routers. *BT Technology Journal*, 24(4):77–83, July 2006.
[73] Janos Komlos and Albert Greenberg. An asymptotically nonadaptive algorithm for conflict resolution in mutliple-access channels. *IEEE Transactions on Information Theory*, IT-31(2):302–306, March 1985.
[74] Sandor Gyori. Signature codeing over multiple access or channel. In *Winter School on Coding and Information Theory*, 2003.
[75] V. Ufimtsev. Construction of a superimposed code using partitions. *Nonlinear Analysis*, 63:e2649–e2653, 2005.
[76] Shakir Abdul-Jabbar and Peter de Laval. Constant weight codes for multiaccess channels without feedback. In *Conference Proceedings on Area Communication, EUROCON 88*, pages 150–153, June 1988.
[77] C. Christian Jonsson. Anti-jamming on the or-channel. In *1994 IEEE International Symposium on Information Theory*, pages 480–480, Trondheim, Norway, July 1994.
[78] http://www.nist.gov/dads/HTML/superimposedCode.html.
[79] P. de Laval and S. Abdu-Jabbar. Decoding of superimposed codes in multiaccess communication. In *Conference Proceedings on Area Communication, EUROCON 88*, pages 154–157, June 1988.
[80] P de Laval. Decoding of superimposed codes. Technical Report LiTH-ISY-R-0958.
[81] P. de Laval. Sliding window reduced search decoding for superimposed codes. Technical report, Linkoping University, 1989.
[82] P. de Laval. Superimposed code reservations systems- description and examples of possible implementation principles. Technical report, Linkoping University, 1989.
[83] P.Z. Fan, M. Darnell, and B.Honary. Superimposed codes for the multiaccess binary adder channel. *IEEE Trans. on Information Theory*, 41(4):1178–1182, July 1995.
[84] S. Chuprun, C. Bergstrom, and B. Fette. Sdr strategies for information warfare and assurance. In *MILCOM 2000. 21st Century Military Communications Conference Proceedings*, volume 2, pages 1219–1223, 2000.
[85] S. Knappe, L. Liew, V. Shah, P. Schwindt, J. Moreland, L. Hollberg, and J. Kitching. A microfabricated atomic clock. *Appl. Phys. Lett.*, 85:1460, 2004.
[86] Goldreich, Goldwasser, Lehman, Ron, and Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.
[87] Danny Harnik and Ran Raz. Monotone complexity by switching lemma. Technical report.
[88] Stasys Jukna. A criterion for monotone circuit complexity. *unpublished manuscript*, 1991.
[89] Stasys Jukna. Combinatorics of monotone computations. *Combinatorica*, 19(1):65–85, 1999.
[90] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require superlogarithmic depth. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 539–550, New York, NY, USA, 1988. ACM Press.
[91] A. A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *COMBINATORICA*, 10(1):81–93, 1990.

[92] Ran Raz and Pierre McKenzie. Separation of the monotone nc hierarchy. *Combinatorica*, 19(3):403–435, 1999.

[93] Ran Raz and Avi Wigderson. Monotone circuits for matching require linear depth. *Journal of the Association for Computing Machines*, 39(3):736–744, July 1992.

[94] Michael Sipser. The history and status of the p versus np question. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 603–618, New York, NY, USA, 1992. ACM Press.

[95] E. Tardos. The gap between monoton and non-monotone circuit complexity is exponential. *COMBINATORICA*, 7(4):141–142, 1987.

[96] L. G. Valiant. Exponential lower bounds for restricted monotone circuits. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 110–117, New York, NY, USA, 1983. ACM Press.

[97] Christy Chatmon, Tri van Le, and Mike Burmester. Secure anonymous rfid authentication protocols. 2006.

[98] Tri van Le, Mike Burmester, and Breno de Medeiros. Universally composable and forward secure rfid authentication and key exchange. In *ACM Symposium on Information, Computer and Communications Security ASIACCS 2007*, March 2007.

[99] A. Krohn. Superimposed radio signals for wireless sensor networks. In *Advances in Pervasive Computing 2006, Adjunct Proceedings of Pervasive 2006*, pages 187–192, May 2006.

[100] A. Krohn, T. Zimmer, M. Beigl, and C. Decker. Collaborative sensing in a retail store using synchronous distributed jam signalling. In *3rd International Conference on Pervasive Computing 2005*, volume 3468.

[101] C. Faloutsos and D. Oard. A survey of information retrieval and filtering methods. Technical Report CS-TR-3514.

[102] W. Frakes. *Information Retrieval: Data Structures & Algortihms*. Prentice Hall, 1992.

[103] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley Professional, 1998.

[104] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.