OSCILLATOR MISMATCH AND JITTER COMPENSATION IN CONCURRENT CODECS

William L. Bahn
Leemon C. Baird III
United States Air Force Academy
Michael D. Collins
Department of Defense

***Abstract*— *The advent of concurrent coding theory means that omnidirectional communication systems can possess a level of keyless jam-resistance comparable to that of traditional spread spectrum systems, all of which rely on shared secret keys. To achieve this, concurrent codecs possess the ability to efficiently separate multiple legitimate codewords that have been superimposed. This is achieved by leveraging a highly asymmetric sensitivity to bit errors and, consequently, a reliance on communication channels having correspondingly high degrees of asymmetry in their bit error probabilities. While suitable physical channels must possess inherently high degrees of asymmetry, this asymmetry can be artificially enhanced using post processing techniques with the effect that system designers can trade small amounts of jam-resistance for increases in noise immunity. Furthermore, to rob potential adversaries of the option of attacking the receiver's ability to synchronize with the transmitted signal, concurrent codecs do not perform real-time adaptive synchronization and instead use asynchronous protocols. To avoid bit misalignments over the length of the packet, such protocols normally require that transmitters and receivers have oscillators with frequency tolerances on the order of one part in ten times the packet length. However, a concurrent codec can use simple post-processing techniques to exploit the asymmetry in bit error sensitivity to give receivers high degrees of immunity to timing jitter as well as high tolerances to oscillator mismatch. This has implications not only for processing gain, but also for implementation cost since transceivers can utilize oscillators having greatly relaxed specifications compared to that required by traditional systems. This paper presents these techniques and analyzes their impact on jam-resistance and oscillator performance requirements.*

I. The Need for Keyless Jam-Resistance

Wireless networks operating in hostile environments require high degrees of jam-resistance to ensure the availability of network resources. The two traditional means of providing this are highly-directional links and spread spectrum. The exclusive use of directional links in highly dynamic mobile ad hoc networks poses practical challenges that virtually guarantee that omnidirectional links will continue to play a significant role in such networks. However, spread spectrum techniques are only as secure as the shared-secrets (i.e., symmetric keys) upon which their jam-resistance is based; this, in turn, is limited by the classic key distribution problem associated with such keys. Therefore a means is needed of ensuring the availability of the channel in large omnidirectional wireless networks that does not rely on symmetric keys.

A seemingly obvious alternative are asymmetric keys since ways of using them to ensure the other classic security goals (i.e., confidentiality, integrity, and authenticity) are well understood. Unfortunately, these all assume and require that data can be successfully exchanged between parties, implying that the communications channel is already available. Consequently, a means in needed of ensuring channel availability in large omnidirectional wireless networks that does not rely on secret keys at all.

This same conclusion can be reached independently by considering a different category of application altogether – namely public-access systems. One example is civilian GPS where communications are one-way and where the pool of authorized users is literally every person on the planet. In this case, secret keys of any kind are precluded since, by definition, hostile parties are authorized users with access to the same keys as everyone else.

For an *unkeyed system* – a system with no secret keys – to be jam-resistant, it must be capable of dealing with multiple overlapping transmissions of legitimate waveforms: some from friendly sources and some from hostile sources. This problem is closely related to the field of superimposed codes; historically, such codes have been limited to applications in which *membership testing* is sufficient since, until recently, no efficient means of fully decoding a superimposed transmission has existed. This situation changed fundamentally with the advent of concurrent codes; by definition, these are superimposed codes that can be efficiently decoded. In fact, although only one realized construction of a practical concurrent code presently exists, its decoding complexity is linear-time with respect to the product of the length and number of transmitted messages.

II. A Brief Review of Concurrent Coding Theory

A detailed explanation of concurrent coding theory is available in [1] and [2] – only a brief summary is provided here. Concurrent codes rest on the decades-old foundation of superimposed codes[3], of which Bloom filters[4] are an example. The heart of a superimposed code is an encoding algorithm whereby a packet containing several different messages can be constructed by performing a bitwise-OR of the codewords

corresponding to the chosen messages. Going the other way, a packet is considered to contain a particular message if it *covers* that message, meaning that all of the marks (i.e., bits whose value is HI) in the codeword for that message are contained in the packet. Ideally, a packet should only cover those messages used to construct it. In practice, this will only be true up to a point; if too many messages are placed into a packet then it will begin covering additional messages. These additional messages go by a variety of different names; here they are referred to as *hallucinations*. In general, parameters in the coding and decoding (i.e., codec) algorithms permit the user to control, at least statistically, how many messages can be contained in a packet before hallucinations start to appear.

Superimposed codes have traditionally found use where it is sufficient to ask if a particular message, or small set of messages, is covered by the a given packet. This is done by performing a "membership test" which involves verifying that all of the marks in the message being tested are covered by the packet. Membership tests can be performed very efficiently; conceptually they involve nothing more than a bitwise-AND between the codeword and the bitwise complement of the packet, with any non-zero result declared a failure.

However, if it is necessary to generate a list of all messages covered by the packet - referred to here as *decoding* the packet - then an exhaustive search spanning the entire message space is generally needed. As recently as 2003, it has been claimed by at least one researcher [5] that no efficient means of decoding arbitrary codewords from very large code books yet exists. Decoding therefore requires an exponential amount of time as a function of the message length. While acceptable for applications having a sufficiently small message space as well as sufficient time and processing power to perform the work, it is totally infeasible given the message spaces, processing capabilities, and time constraints found in typical communication systems.

Thus, for communication systems, a subset of superimposed codes is needed in which the entire list of messages can be extracted from a packet in an "efficient" manner. This requirement is the defining characteristic of a concurrent code compared to the broader set of superimposed codes.

III. A Review of the BBC Encoding and Decoding Algorithms

The BBC algorithms are a pair of algorithms that encode and decode data to and from concurrent message packets. Like concurrent coding theory itself, extensive details about them and their behavior are now in the open literature[1], [2] and only a brief description is provided here. The underlying basis for the BBC algorithms is that m-bits of data are first transformed into a message which is then encoded, one bit at a time, using progressively longer prefixes to construct a c-bit codeword. This allows for the efficient decoding of the packet since messages can be recovered one bit at a time by looking for progressively longer message prefixes. Finally, the data can be extracted from the recovered message.

The BBC encoding algorithm is given in Algorithm 1.

Algorithm 1 BBC Encode(D)

This function takes m-bits of data, pads it with k checksum bits at locations specified by the protocol in use to produce the message M, and then places marks in the codeword P at locations determined by hashes of all possible prefixes of the padded message. In addition, "bookend" marks are placed in the packet at the first and last locations

M = D with k spaces added as specified by the protocol.
$P \leftarrow 0$ (i.e., P is a c-bit vector initialized to all spaces)
Place bookend marks in the packet.
for ($n \leftarrow 1 \ldots m + k$ *($m + k = length(M)$)*) **do**
  Compute $L = Hash(M[1 \ldots n])$ such that $1 \leq L \leq c$
  Place a mark in P at location L
end for
return P

NOTES:
1) $M[1 \ldots n]$ *is the* n*-bit prefix of* M.
2) $1 \leq Hash(X) \leq c$.

In practice, decoding a packet is most efficiently done using a depth-first search of the message space; however, it is easiest to describe in terms of a breadth first search, as depicted in Algorithm 2.

The dominant parameter in the algorithm is the expansion factor, e, which is the ratio of the length of the codeword, c, to the length of the original data, m (i.e., $c = me$). The expansion factor determines how the output of the hash function is interpreted; specifically, it is interpreted as being one of the c possible locations in the codeword. The expansion factor is roughly equivalent to the processing gain of a traditional spread spectrum system, both in terms of bandwidth spreading and in terms of the degree of jam-resistance.

The role of the checksum bits is to kill hallucinations. They do this by inserting k marks into the codeword at locations that are functions of the entire message. Since the decoder knows a priori the value of all checksum bits, the list of covered prefixes cannot grow when decoding such a bit. Furthermore, hallucinations in the list survive the decoding of a checksum bit purely by chance. In general, the fraction of hallucinations that survive each checksum bit is equal to the packet's mark density, hence checksum bits exterminate hallucinations exponentially.

One important point to note about concurrent codes – and true of all superimposed codes – is that any one codeword is relatively sparse, consisting almost entirely of spaces with

Algorithm 2 BBC Decode(P)

Whenever a possible packet is identified, based on finding a pair of "bookend" marks, then all of the data in the packet is extracted by first identifying all messages covered by the packet and then extracting the data from each covered message. At any given stage of the process, a list of message prefixes that are covered by the packet is maintained. The prefixes are lengthened one bit at a time and only those still covered are retained. If the next message bit is a checksum bit, then the prefix extended with a '0' bit is considered. However, if the next message is a data bit then the prefix extended with a '1' bit is also considered.

if Bookend marks are found (marks at first and last bit position) **then**
  $M \leftarrow \{\}$ (i.e., M is an empty message list)
  for $(n \leftarrow 1 \ldots (m+k))$ **do**
    Expand partial messages in list to n-bits
    for (Each partial message, M_j, in M) **do**
      Remove M_j from list M
      Add $M_j : 0$ to list M (i.e., append a '0')
      if n corresponds to a data bit **then**
        Add $M_j : 1$ to list M (i.e., append a '1')
      end if
    end for
    Prune messages from list
    for (Each partial message, M_j, in M) **do**
      Compute $L = Hash(M_j)$
      if P does not contain a mark at location L **then**
        Remove M_j from list M
      end if
    end for
  end for
end if
Extract the data from the messages
for (Each message, M_j, in M) **do**
  Strip all checksum bits from M_j
end for

a few pseudorandomly scattered marks. The fraction of a codeword (or packet), that consists of marks is referred to as the *mark density*; this plays a key role in determining decoder performance.

IV. The Physical OR-channel

Concurrent codes, like their superimposed brethren, require an OR-channel if they are to retain necessary properties. An OR-channel is one in which all of the codewords present in a channel are combined via a bitwise-OR operation. From a practical standpoint, this means that the receiver should produce a mark whenever any of the transmitters are broadcasting a mark and should produce a space only when all of the transmitters are broadcasting a space. This is difficult to achieve with most modern binary modulation schemes because they are designed to produce symmetric channels having nearly identical bit error probabilities, regardless of whether a mark or space is transmitted, as this minimizes the overall bit error rate (BER). However, by using a highly asymmetric channel, such as old-fashioned On-Off Keying (OOK), an OR-channel can be implemented with a high degree of fidelity by setting the detection threshold sufficiently low. Since such a channel is not symmetric, it cannot be characterized by a single error rate. Instead, it has a mark error rate (MER) and a space error rate (SER). The MER is the probability that a space will be received even though one or more marks were transmitted and the SER is the probability that a mark will be received even though no marks were transmitted. In general, lowering the detection threshold will lower the MER at the expense of raising the SER.

V. Jam-resistant Nature of Concurrent Codes in and OR-channel

We can now discuss how concurrent codecs provide high degrees of assurance that the channel will remain available even in the presence of considerable hostile jamming. If a sender transmits a message that is encoded with a concurrent codec, then as long as all transmitted narks are received, the message will be contained in the list of messages produced by the receiving codec. By contrast, successful reception of all spaces is not necessary; in fact, a significant fraction can be received erroneously without significant impact on decoder performance. The attacker thus has two options available to them: (1) they can try to force the receiver to make a mark error, or (2) they can try to flood the receiver with space errors (false marks).

The first option, while highly attractive since the decoder is extremely sensitive to mark errors, is not very practical because once the sender transmits energy into the spectrum, the attacker cannot easily remove that energy; they can add to it or corrupt it, but they cannot easily diminish it in any practically meaningful sense. The receiver, on the other hand, is not relying on being able to extract any information from the energy that was transmitted beyond the mere detection of its existence. If the receiver's detection threshold is sufficiently low, detection is virtually guaranteed. Furthermore, promising methods of tolerating modest mark error rates are presently being explored.

This leaves the second option which, while not particularly attractive since the decoder is highly insensitive to space errors, is never-the-less always possible. The hostile party can force a sufficiently high SER placing the packet beyond the codec's ability to decode with available resources. However, doing so requires the attacker to expend considerably more energy, all

else being equal, than the genuine sender did thus exposing them to the network defenders. In order to successfully jam the channel the attacker must accept a commensurate level of risk. If that risk can be made unacceptable then the channel will most likely remain available.

It should be emphasized that concurrent codecs promise neither jam-proof communications nor a level of jam-resistance greater than that of uncompromised spread spectrum. What concurrent codecs provide is a comparable level of jam-resistance without the need for symmetric keys and the corresponding key management nightmare. Thus, from a practical perspective, concurrent codes can be viewed not as a means of improving jam-resistance, but as a tool for improving key management.

VI. Attack Model and Codec Configuration for this Discussion

As with traditional spread spectrum systems, the jam-resistant properties of a concurrent codec can be tailored by choosing various parameters accordingly to the threat model in use. For the purposes of this discussion, we will assume a threat model that permits the attacker to drive the packet mark density as high as 33% before attracting unacceptably high levels of interest from the network defenders. At this density the receiver workload is doubled, meaning that it must process one hallucination for every intentional message in the channel.

Somewhat arbitrarily – and except as indicated otherwise – we will assume that the codec being considered has been configured so that each message contains 1000 bits of data appended with 32 checksum bits. We will further assume that the codec has an expansion ratio of 1000, meaning that each codeword is one million bits long. Note that this is not a hypothetical configuration, it is one of the configurations frequently used in the present software-defined radio RF demonstrator. This system will be discussed in more detail later.

With this attack model and codec configuration, the attacker needs to transmit approximately 320 times the energy (25dB) as the legitimate sender to reach their risk limit. Owing to collisions in the hash function, they can transmit, on average, almost 400 randomly chosen attack messages. Due to the combined impact of the attack messages and the hallucinations they create, this will force about 800 times the computational workload on the receiver compared to if there were no attack.

VII. Oscillator Requirements in Truly Asynchronous Protocols

An asynchronous communications protocol is generally considered to be one in which any timing information needed to receive and decode the signal is embedded in the signal itself, as opposed to having a separate synchronizing signal. Most modern asynchronous protocols embed sufficient timing information to permit the receiver to detect the transmitter's clock rate and to adjust its own reception clock to match. Many such embedded-clock schemes exist, most of which use coding algorithms that result in edge-rich symbols. These edges are easy for the receiver to detect and are, for example, used to control a phase-locked loop (PLL) so that the receiver's clock is kept in tight synchronization with the clock used to transmit the data.

Unfortunately, having the receiver use the received signal to control its own oscillator provides an obvious target for hostile parties wishing to disrupt the communications link; if they can interfere with the synchronization process, which may only require a relatively small amount of transmitted energy, then they can jam the channel. This vulnerability can be removed if a truly asynchronous system is used in which the receiver's oscillator is free running and no attempt is made to servo it to the transmitter. An example where this is typically the case is RS-232; in this protocol the only timing information that is transmitted is the start of the packet. The receiver attempts to detect this packet start and, once detected, relies on its oscillator being sufficiently matched to that of the transmitter to allow it to capture the symbols properly over the course of the packet.

Since both the transmitter and receiver oscillators are free running and no attempt is made to servo the latter to the former, they must be sufficiently close to each other naturally to prevent *framing errors*. A framing error occurs when, among other causes, the accumulated mismatch between the two oscillators is sufficient to cause the receiver to be misaligned with the transmitted signal. A useful rule of thumb is that the accuracy of the two oscillators must be at least an order of magnitude better than one part in the length of the packet. For instance, in RS-232 where packets rarely exceed a dozen bits, oscillators whose accuracy is on the order of 1% are needed. With typical crystal oscillators exceeding this by a couple orders of magnitude, this is rarely a concern for RS-232 communication links. However, if the packet length is to be a million bits, as is the case with the codec being discussed, then oscillators with accuracies on the order of 0.1ppm (10^{-7}) are needed. While such oscillators exist – state of the art is about 10^{-10} – they are expensive and frequently difficult to incorporate into designs suitable for harsh field conditions.

Anecdotal proof that concurrent codecs enable truly asynchronous communications using oscillators having tolerances a few orders of magnitude worse that what would be expected is borne out by the fact that one million bit packets are being successfully used in an RF demonstrator having oscillators rated at 100ppm. From a crude perspective, this means that by the time the receiver reaches the end of the packet it would not be surprising for the misalignment to exceed well over one hundred bit positions. While actual experience shows that the misalignment seldom exceeds fifty bit positions, even this is two orders of magnitude greater than the half-a-bit needed to cause, with highly likelihood, a framing error.

None-the-less, the RF demonstrator has proven to be very robust despite the relatively high oscillator mismatch because the receiving codec performs both oscillator mismatch and jitter compensation. This is possible because of the high asymmetry in the sensitivities to mark and space errors; just as the jammer can force a large number of space errors before having a significant impact on the receiver's workload, the decoding algorithm has wide latitude in assuming where mark errors exist while maintaining a similarly benign impact.

VIII. The Demonstration Platforms

Two platforms were used to examine oscillator mismatch and jitter effects and to develop compensation strategies. The first was an audio demonstrator whose physical layer used the internal microphone and sound cards in laptop computers. The second was an RF demonstrator using software defined radios (SDR) for the physical layer. In both cases the encoded packet was transmitted using OOK (On-Off Keying) of a pure tone (a.k.a., CW).

For the audio demonstrator, a typical configuration involved 128 bits of data and 32 checksum bits encoded into a 6,400 bit codeword (expansion factor of 50). The packet was transmitted from the laptop's speaker at 1,000 baud using a 4.4 kHz sinewave. The receiving laptop sampled its microphone at 11.025 kSa/s.

For the RF demonstrator, a typical configuration was 1,000 bits of data and 32 checksum bits encoded into a 1,000,000 bit packet (expansion factor of 1,000). The packet was transmitted at one megabaud using an unoccupied region of the 915MHz or 2.45GHz ISM bands. After down-converting from the carrier, the receiver sampled the baseband signal at 4MSa/s.

With either demonstrator, three platforms were generally used: two transmitters and one receiver. One transmitter was "genuine" sender and generally sent packets containing one or, at most, a few messages. The second transmitter was the "jammer" and generally sent packets containing many messages and/or random pulses. The jammer, when used, was always at least as close to the receiver as the legitimate sender – typically midway between them – and broadcasting at least as strong a signal as the genuine sender.

Even though the audio demonstrator's received digitized waveform was the complete signal at the carrier frequency while the RF demonstrator's waveform was a baseband signal, both platforms used identical signal processing chains on the received waveform. The first step employed a software radiometer. The signal was squared to produce a signal proportional to the instantaneous power and then low-pass filtered using a first-order infinite-impulse-response (IIR) filter. The filter output was converted to a binary signal using a software Schmidt-trigger discriminator. The signal at this point, which was still at the sampled signal rate, was decimated to produce the data stream sent to the decoder using a retriggerable one-shot whose output was sampled at the baud rate. In order to improve the real-time performance of the codec, all of the above steps were integrated into a single tight processing loop termed a *decimating radiometric discriminator*.

The decoder was presented with a continuous binary data stream at the baud rate. Every mark in the stream was presumed to be the leading mark of a packet. If, in fact, there was no packet starting at that point, then the decoding tree dies almost immediately. Hence there is a small, but non-zero, decoding overhead associated with any non-zero packet mark density.

IX. Jitter Compensation

A number of error sources exist that can, at least informally, be lumped together as being equivalent to timing-jitter. There is, of course, actual edge jitter in both the transmitting and receiving oscillators and circuitry. Furthermore, particularly if the receiver is radiometer based, there are time-walk issues. Time-walk occurs because the exact time at which the discriminator threshold is crossed depends on the amount (and the profile) of energy in the signal. Pulses with just enough energy to cross the discriminator's threshold will generally result in a delayed response going high and a quicker response going low than pulses with greater energy. At least for the rising edge, this can be greatly alleviated by using more sophisticated circuits such as constant-fraction discriminators. However, for simplicity, we will assume that such techniques are not being used and that, in general, the discriminator output suffers energy-dependent time-walk. In general, we will assume that there is always some uncertainty about whether a detected mark is actually in the proper bit position.

Traditionally, combating jitter via coding is difficult since correcting one error is likely to create another. However, concurrent codecs tolerate jitter extremely well because of the very high asymmetry in bit error sensitivities. As a result, techniques that reduce the likelihood of experiencing a mark error can be effectively utilized even if they produce significant space errors. Two techniques are presented here: mark extension and decoder windowing. Both perform essentially the same task, but how they do it – and the advantages and disadvantages of each – are different.

Mark extension refers to actually increasing the length of the marks in the radiometer output. This can be done in the transmitter and/or the receiver. The transmitter can simply increase the duration of each mark it transmits. Alternatively, the encoder may place marks at J adjacent locations in the codeword starting with the nominal single-mark location. To date, experiments with the audio demonstrator have never needed a value of J greater than four while the RF demonstrator has never shown any benefit from any J greater than two. The receiver can extend the marks by slowing the recovery time of the radiometer or by extending the length of the one-shot. The principal advantage of transmitter-side mark extension is that

only the transmitter's marks are extended; marks due to noise or to the attacker are not. Conversely, if the receiver extends the marks there is a significantly greater impact on jam-resistance since all of the marks, regardless of source, are extended. The main disadvantage of transmitter-side mark extension is that the transmitter must know how wide to make the marks. Purely receiver-side techniques are preferred, where feasible, to avoid having to negotiate anti-jamming parameters over a channel that is being jammed.

Decoder windowing also has the ability to compensate for jitter by virtually extending the marks in the packet. Conceptually, this is very similar to how the transmitter extends marks; namely, if the hash function indicates that a mark should be at a particular location, then the decoder considers it found as long as any mark is within a window that extends J bits beyond the nominal location. Decoder windowing has the advantage of being easily implemented in hardware, but it lacks the sub-bit resolution attainable when performed in either the transmitter or the receiver.

X. Oscillator Mismatch Compensation

The original technical report[1] assumed that a truly asynchronous protocol would require time bases that were accurate – or at least relatively precise – to within approximately one part in ten times the packet length (the common rule-of-thumb). It also noted that oscillators suitable for small battery powered devices with accuracies on the order of 10^{10} existed, at least in the laboratory.

Subsequent practical experience with the audio demonstrator has shown that much worse accuracies can be tolerated with only minor impact on receiver workload. Originally all of the computers were identical models purchased as part of the same contract; thus the oscillators use by the sound cards were probably unrealistically well matched since they were used in identical circuits and probably came from the same lot, if not the same silicon. However, eventually one of the computers was replaced by a different make and model; it was unable to successfully send or receive packets to the other two machines. When the transmitted and received waveform files were examined side-by-side it was discovered that the received waveform was 41ms shorter than the transmitted waveform. This implied that the time bases of the two platforms differed by approximately 0.64% (6400ppm). Using the rule-of-thumb mentioned earlier, this would limit packet lengths to approximately sixteen bits. However, since the entire packet must be stored in memory, the highly asymmetric error sensitivities immediately offered a potential solution.

Though unconventional, an oscillator may be thought of as a processing block that converts time to cycles, characterized by a certain transfer function, namely how many cycles it produces at the output per unit of time supplied at the input. From this perspective, two mismatched oscillators simply have a relative

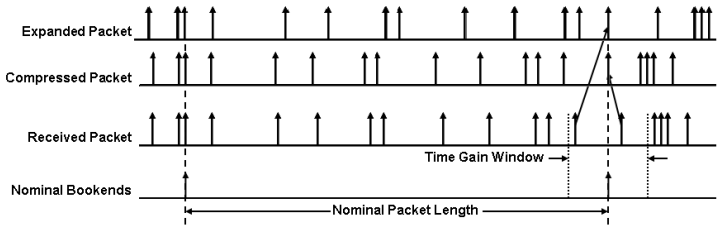

Fig. 1. Time-gain error correction example.

gain error in their transfer functions. Because gain errors are common in many applications (e.g., fixed-pattern noise in solid state imagers), gain correction is routinely performed. One common technique is to calibrate the system using a two-point correction. To do this, two reference signals are measured with the uncorrected device: one a low level signal and the other a high level signal. From those two points and their assumed correct values, gain (and offset, which is not important here) correction factors can be computed and applied to all of the data received by that device. This approach is very simple and works extremely well provided the detector is linear.

Applying a two-point correction to the problem posed by mismatched oscillators led to the solution of simply embedding a set of reference signals that could be used to calibrate a correction coefficient. At the time this problem was first identified the packet already contained a start-of-packet mark used for decoder convenience. The addition of an end-of-packet mark (thus creating a pair of "bookend" marks) was the obvious choice since it placed one reference signal at a low value of time and the other at a high value. The process used is illustrated in Figure 1. In this example, after decoding any packets that start with the second mark from the left, the decoder advances down the received bit stream until it encounters the next mark. It treats this as a potential start-of-packet mark and computes where in the bitstream the end-of-packet mark should be. However, instead of just looking at that single location, the decoder considers any mark appearing within a window, centered on that location, as a potential end-of-packet mark. If N such marks are found within the window, then the decoder performs N complete decodes of the bitstream compensating for the apparent time-gain error associated with each candidate end-of-packet mark by expanding or compressing the packet as necessary.

Returning to the case of the new laptop computer for the audio demonstrator described above, the new version of the decoder was configured to permit up to $\pm 1\%$ of oscillator mismatch. As a result, the three platforms immediately began communicating with each other with no noticeable impact on performance. (Certainly a performance impact existed, it was just sufficiently small so as to escape casual notice.)

In the case of the RF demonstrator, the rated oscillator accuracies of 100ppm (each) would normally limit the packet

length to something on the order of 1,000 bits instead of the 1,000,000 bits generally used. Despite this, the receiver worked well even without any time-gain correction. This was believed to be the result of two things: (1) The software defined radios in use were all ordered at the same time and have identical circuitry and use identical crystal oscillators that likely came from the same production lot, and (2) the 1-bit wide jitter compensation in use was sufficient to compensate for what minor time-gain error existed.

To estimate the additional workload impact that time-gain compensation imparts, consider that if no message exists in a packet that the expected number of calls to the hash function, N_h, (even for infinitely long messages) is only

$$N_h = \frac{2}{1-2\mu} \tag{1}$$

where μ is the packet mark density. At the limit of the threat model (i.e., where $\mu = 0.33$), this means that the expected number of calls is only six. However, this workload penalty must be paid for each candidate end-of-packet mark in the time-gain window. If this window is $\pm\epsilon$ of the packet length, then it would be expected that $2\mu\epsilon me$ candidate marks will be found. Thus the total expected burden, per mark in the bit stream, is

$$\frac{4\mu\epsilon}{1-2\mu} me \tag{2}$$

For the RF demonstrator's typical configuration, the expected 67 candidates would equate to 400 hash calls at the threat model limit when $\epsilon = 100ppm$ (10^{-4}). Even under ideal circumstances with no attack, decoding a packet containing a single message involves a minimum of $2m+k$ calls to the hash function. For the assumed configuration this is 2,032 hash calls. At the threat limit this doubles to slightly over 4,000 hash calls for every actual message, be it from the genuine sender or the attacker.

It must be kept in mind, however, that the time-gain burden applies to each mark in the bit stream. Hence, assuming that legitimate packets have been transmitted back-to-back, the total number of hash calls due to this burden, at the edge of the threat model, would be approximately 133 million. To place this in perspective, decoding a packet that has been driven to this density (via the addition of approximately 400 attack messages) would require, on average, 1.6 million calls to the hash function. Thus the time-gain correction burden would increase the receiver workload by a factor of about 83. However, this cost allows the use of an oscillator that is three orders of magnitude less accurate than would otherwise be required. It must also be noted that this is the penalty paid at the edge of the threat model. If, for example, the attacker is transmitting 100 attack messages (and, hence 100 times the energy as the genuine sender) bringing the packet density to approximately 10%, the burden is only about 25.

A reasonable estimate of the burden, per message, is

$$\frac{2\mu\epsilon}{1-\mu} me \tag{3}$$

where it is assumed that the packet density has been driven to the level it is at solely through the presence of messages in the channel. The simplifications made to obtain this expression are conservative in that the actual burden will be less.

Preliminary measurements with the RF demonstrator show that even using $\epsilon = 0.1\%$ (i.e., allowing oscillators that are four orders of magnitude worse than otherwise required) the decoder is capable of keeping up with the additional burden quite well, especially if a small number of checksum bits are prepended to the data bits when the message is formed so as to extinguish empty packets faster.

XI. Conclusion

To rob an adversary of as many attack modes as possible, a concurrent codec can use simple modulation (OOK) and reception (radiometry) techniques that do not incorporate sophisticated timing recovery schemes. Such asynchronous protocols, when used with traditional modulation techniques and coding schemes in a symmetric channel, result in unacceptably high bit error rates for useful packet lengths. However, the highly asymmetric bit error rates needed for the use of a concurrent codec opens the door for techniques that leverage that asymmetry allowing the designer to trade processing effort against component tolerances and circuit performance specifications, particularly with regards to jitter and oscillator mismatch. Given that a concurrent codec lends itself to highly parallelized processing algorithms and that processing capacity in general keeps getting faster and cheaper, this is an exchange well worth considering.

XII. Acknowledgements

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

References

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

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

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

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

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