Practical Error Correction for Resource-Constrained Wireless Networks: Unlocking the Full Power of the CRC

Travis Mandel
Computer Science and Engineering Department
University of Washington
tmandel@cs.washington.edu

Jens Mache
Mathematical Sciences Department
Lewis & Clark College
jmache@lclark.edu

ABSTRACT

Bit errors are common in wireless networks, and techniques for overcoming them traditionally consist of expensive retransmission (e.g. Automatic Repeat reQuest (ARQ)) or expensive Forward Error Correction (FEC), both of which are undesirable in resource-constrained wireless networks such as wireless sensor networks (WSNs). In this paper, we present TVA (Transmit-Verify-Acknowledge), a protocol that can correct errors without adding additional redundancy to data packets. Instead, TVA corrects errors using the redundancy inherent in Cyclic Redundancy Checks (CRCs). The ubiquity of CRCs has the advantage of allowing TVA to be both backwards-compatible and backwards-efficient with link-layer protocols such as IEEE 802.15.4. We present a novel method of CRC error correction, which is compact and computationally efficient, and is designed to correct the most common error patterns observed in WSNs. We demonstrate that TVA provides reliability effectively equivalent to that of ARQ. We perform trace-driven simulations using data from sensor network deployments in different environments and analyze TVA's performance at different message lengths. To demonstrate the practicality of TVA, we implement it in TinyOS, and perform experiments on MicaZ motes to evaluate TVA in the presence of 802.11 interference. We find that TVA improves over ARQ and FEC-based protocols, using 31% less redundant communication and 30% less additional time to recover errored packets compared to ARQ.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*; E.4 [**Coding and Information Theory**]: Error Control Codes

General Terms

Design, Experimentation, Measurement, Theory

Keywords

Wireless, IEEE 802.15.4, Cyclic Redundancy Check (CRC), Error Correction, ARQ, Performance Evaluation

1. INTRODUCTION

In wireless networks, physical obstructions and 802.11 interference are two common causes of packet corruption in the form of bit errors. Since reliability is paramount, Automatic Repeat reQuest (ARQ) is typically used to ensure successful packet delivery in the face of errors. These expensive resends can cause a reduction in throughput and energy efficiency, which is particularly critical in resource-constrained networks such as sensor networks. To reduce retransmissions, Forward Error Correction (FEC) has been proposed [12, 21, 9], which adds additional upfront redundancy to each packet, requiring more resources to compute and more bandwidth to transmit. In high-error scenarios the use of FEC is clearly desirable, but it adds unnecessary redundancy in low-error scenarios. In this case, one would ideally refrain from adding additional redundancy to packets without errors, but when an error occurs, correct the packet efficiently with a minimum of additional communication.

We introduce Transmit-Verify-Acknowledge (TVA), a new link-layer protocol that reliably corrects a large percentage of errors without requiring any part of the original message to be resent or adding additional redundancy to data packets. TVA is backwards-compatible with 802.15.4, requires no hardware modifications, and, in the absence of bit errors, acts like a typical ARQ protocol. But for the most commonly-occurring bit errors, TVA corrects the message and uses only a small amount of additional communication to verify the correction.

TVA is able to correct messages by exploiting the Cyclic Redundancy Check (CRC), a code appended to almost all modern network packets. The CRC is generally used solely to determine if the message was received correctly. However, we show that there is enough redundancy in this checksum to correct a large percentage of bit errors that occur in practice.

To see how TVA works, consider the following Alice and Bob scenario. Alice sends the original message over a wireless channel to Bob. During transmission, the message is corrupted with a bit error (perhaps due to wireless interference, or physical obstructions). When Bob receives the message, he first checks the CRC to determine if the message is correct. Since it is not, Bob attempts to correct the message using our novel CRC error correction procedure (section 3.2). In certain cases when the packet corruption is overwhelming, the error correction procedure may fail, in which case Bob drops the packet and waits for a resend. If the correction succeeds, Bob sends Alice a packet containing a short description of the received message (see details in section 5). Alice then checks that the description matches the original packet. In the rare event that the description does not match, indicating that the corruption of the original packet was too severe, Alice resends the original message. Otherwise, Alice sends Bob a very short confirmation message, and Bob then passes the corrected original message up the stack.

TVA's key contribution is a novel method of recovering from bit errors that is practical for resourced-constrained devices. This is different from theoretical work in error correction (e.g. LDPC [18]) which have guarantees about optimizing channel usage but, unlike TVA, are not backwards-compatible, and expensive to implement on devices with severely constrained memory and computational abilities. Other approaches such as Partial Packet Recovery (PPR) [8], require "hints" from the radio hardware to achieve good error correction ability, which, in contrast to TVA, requires extensive modification to hardware and thus is unsuitable for deployment in existing networks. Block-level retransmission schemes such as Maranello [6] do no error correction at all but rather require extra redundancy to locate the errored data section and resend it, and thus generally have been shown to be worse than FEC techniques [12]. In contrast, TVA is able to locate errors without any additional redundancy, allowing it to outperform both FEC and ARQ.

TVA has the following key features:

- TVA is backwards-efficient with 802.15.4 (see section 8.4). In particular, this means that energy will not be wasted sending extra redundancy or special packets unless the receiver can utilize them.
- TVA is highly efficient, both in memory consumption and in CPU usage. This is in contrast to previous memory-intensive methods of CRC error correction [17]. See section 8.2.4 for more details on how we implement the required buffering in a memory-efficient fashion.
- TVA is effectively as reliable as ARQ even in high-error situations, despite further exploiting the CRC (see Section 5.3).

Since TVA is especially well-suited to WSNs, we have implemented TVA on Crossbow MicaZ sensor motes, which have only 4KB RAM and a 8MHz processor. We evaluated TVA first on a trace-driven simulation using traces collected by others, and then evaluate TVA on a sensor testbed. Our results reveal the following:

- In simulation, TVA performs better in average code rate and throughput than ARQ and FEC-based schemes in a variety of environments.
- On high-error trace sets, TVA improves throughput by as much as 3x over ARQ.
- In our MicaZ experiments, TVA tends to outperform both ARQ and FEC-based schemes in terms of throughput and average code rate.
- In our MicaZ experiments, TVA uses an average of 31% less redundant communication and 30% less additional time to recover errored packets compared to ARQ.

The paper makes four main contributions:

- We present a novel method of CRC error correction which is highly efficient in both memory and computation time,
- We identify the most common errors observed on our hardware and show how our error correction method can be tailored to correct them (though the technique generalizes to many other types of error patterns),
- We show that one can use the CRC for error correction without a loss of reliability by using a short, carefully tailored confirmation exchange,
- We show that TVA is an attractive, practical choice for low-error environments.

2. BACKGROUND AND RELATED WORK

We first discuss background of two areas with which the reader may be unfamiliar, CRCs and Hybrid ARQ, and then survey closely related work.

2.1 Cyclic Redundancy Checks (CRCs)

CRCs use binary polynomial division by a generator polynomial G (of degree n) to detect errors. A good description of CRC calculation can be found in [18]. One critical point is that for an uncorrupted message (with appended CRC), the CRC computed by the receiver will be zero.

2.2 HARQ (Hybrid ARQ)

Hybrid ARQ is an attempt to introduce Forward Error Correction (FEC) to the ARQ protocol. In HARQ type I, extra FEC bytes (using some error correction code) are appended to each data packet. If corrupted data is received, correction is attempted. If the correction succeeds, an ACK is sent, otherwise the message is discarded [14]. HARQ is a good point for comparison, because it robustly adds an error correction scheme to ARQ. When comparing versions of HARQ using different amounts of redundancy, we use the notation HARQ-t, where t is the number of redundant bytes.

One popular FEC scheme that can be used with HARQ is Reed-Solomon (RS) coding, which is particularly effective at correcting burst errors. Any t redundant bytes of data can correct all errors in t/2 bytes [10]. However, RS codes can only detect t errored bytes [10], so for maximum reliability (and for compatibility reasons) they must be usually combined with a CRC. The CRC is then checked first, only if it indicates corruption is the more expensive Reed-Solomon decoding done.

2.3 Related Work

Related work falls into two areas: link-layer protocols that attempt to improve wireless efficiency in the face of bit errors, and previous work in CRC error correction.

2.3.1 *Link-layer protocols for efficient error recovery*

There have been several investigations into using simple FEC techniques to correct corrupted packets in WSNs, such as linear block codes [9] or triple redundancy [21]. Although these codes show some improvement over ARQ, they require substantial redundancy for relatively little error correction ability.

In [12], Liang et al. present TinyRS, a full-featured Reed-Solomon library for TinyOS, which the authors claim is optimized to allow powerful Reed-Solomon error correction in WSNs. They find that TinyRS can correct a high percentage of errors due to wireless interference, and thus recommend it, in combination with other techniques, to improve throughput in WSNs. As such, we compare ourselves to TinyRS in our experiments (sections 8.3 and 9).

LDPC [18] is an error correction code which is gaining popularity due to its strong detection and correction capabilities, and as such does not necessarily need an additional detection code (such as a CRC) to guarantee reliability, even for relatively short LDPC codes. Choi and Moon in [2] evaluated the practicality of LDPC codes in WSNs. They found that, despite using the latest LDPC optimizations, implementing LDPC on MicaZ motes required one to increase the RAM of the motes by 800%.

Partial Packet Recovery (PPR), as presented in [8], is a scheme with a similar goal, to reduce retransmission without adding additional redundancy to data packets. PPR relies on exploiting the hardware's demodulation of the signal to get confidence estimates for individual pieces of the signal, and request retransmission of only the pieces that are likely in error. However, PPR requires significant hardware modification, and as such is not suitable for current use.

Maranello [6] is a 802.11 link layer protocol which, like TVA, does not require hardware modification or additional upfront redundancy. Unlike TVA, however, Maranello does not use the CRC to locate errors, so it must send back very long NACK messages describing the received packet in order for the sender to determine which corrupted blocks to resend. The size of the NACK message is a major drawback to Maranello, and follow-up studies [5] showed that for this reason Maranello performs worse than block-based ARQ schemes (which add significant amounts of upfront redundancy). In turn, studies indicate that FEC such as Reed-Solomon is preferable to block-based ARQ schemes in WSNs [12], and hence we feel these FEC-based protocols are the best point of comparison for TVA.

2.3.2 CRC error correction

Relatively little work has been done in using CRCs for the correction of bit errors. In fact, it is a widespread misconception that CRCs are capable only of detection, and not correction. For example, a 2009 article from IEEE Transactions on Computers [1] writes, *"However, since the CRC codes cannot correct errors, a retransmission is needed when the corruption has occurred in the data. The inability to correct errors sometimes becomes a critical problem when the data integrity is essential but the retransmission is impossible or very costly."* While basic theory textbooks such as [18] note that CRCs are linear codes, and thus theoretically capable of error correction, investigations into the practicality of exploiting this capability have been limited thus far.

McDaniel's work on single-bit error correction [17] gives a preliminary analysis of using CRCs to correct single bit errors. In contrast to our flexible and memory-efficient approach, he uses a memory-intensive table-based approach over a fixed message length, and claims that CRC error correction is impractical with standard generator polynomials. We describe how to correct complex errors, even with standard polynomials.

ATM (Asynchronous Transfer Mode) is a commercial protocol that uses CRC error correction in practice [13]. However, it uses an 8-bit CRC to correct only single-bit errors over a fixed header length, unlike our scheme which is concerned with correcting multiple-bit errors over all message data, including the variable-length payload.

Our preliminary investigations of CRC error correction, [16] and [15], provide some evidence that CRC error correction can be useful in wireless networks. These theoretical papers show that standard CRC generator polynomials are capable of correcting not only single-bit errors, but bursts and double-bit errors as well, and analyze the effect the choice of CRC polynomial has on correction capability across messages of varying lengths. However, this early work provides no evaluation of CRC error correction in practice, assumes that errors are distributed as bursts, makes unrealistic assumptions about hardware CRC calculation (see section 8.2.1), and presents a memory-intensive correction algorithm unsuitable for sensor networks.

3. THE THEORY OF CRC ERROR CORRECTION

Traditionally, CRC error correction has been limited to correcting only a single bit of data. Limited work [15] has also considered burst errors. Bursts and bits, however, are just special cases of error patterns. For our purposes an error pattern is defined as a set of index lists (called error sequences), where each index list corresponds to the indices of the bit errors in a single erroneous message. In order to have a finite set of error sequences, we require a maximum message length X. Below we describe error correction in this more general setting.

3.1 Tabular Method

3.1.1 Theoretical Justification

CRCs are a linear code [18]. This means that for any m_a, m_b, $CRC(m_a) + CRC(m_b) = CRC(m_a + m_b)$. Since CRCs operate over a binary finite field, addition is equivalent to the binary XOR operation. This means $CRC(m_a) \oplus CRC(e) = CRC(m_a \oplus e)$, where e is the error pattern. Now let m_1 be a correct message, so it should have a valid CRC attached and thus we know (see section 2.1) that $CRC(m_1) = 0$. So $CRC(m_1 \oplus e) = CRC(m_1) \oplus CRC(e) = CRC(e)$. Thus, if we know $CRC(e_i)$ for all e_i, then given the CRC of a corrupted message we can search through the error pattern CRCs for a matching CRC, knowing that the CRC of the true error pattern e will match.

If we can find an error sequence e' such that $CRC(e') = CRC(e)$, XORing the error sequence it represents into the message will cause the resulting message CRC to be 0. To see why, if we assume $m_2 = m_1 \oplus e$ then $CRC(m_2) = CRC(e) = CRC(e')$, and

$$CRC(m_2 \oplus e') = CRC(m_2) \oplus CRC(e') = 0$$

This means, if $e' \neq e$, the correction procedure corrupts the message further, and we will be unable to detect the corruption by using the CRC. Section 5 addresses this problem.

3.1.2 Description

All previous work on CRC error correction that we know of has employed the tabular method [17, 13, 15]. The first step in this method is to precompute an error correction table T. In order to construct T, one must in effect simulate the error pattern P and record a mapping of the resulting CRC to the original erroneous bits. Let our maximum data length be $X < 2^n - n$. We then simulate all possible error sequences of our pattern P over messages of length $X + n$, creating i error sequences z_i which indicate errored bits by a 1 and non-errored bits by a 0. We then fill our table T such that $T[CRC_G(z_i)] = a_i$, where a_i is an error sequence identifier[1]. If no two mutations of Z have the same CRC, meaning CRC_G is injective for every z_i, then G is called a *valid* polynomial for error correction. This means given any message M' (consisting of an original message M of size X with an n-bit checksum C_1 appended), one can correct any of the error sequences contained in the pattern P.

To correct errors using the tabular method, we must normally ensure that the sender has padded the message M to the maximum message length X when computing C_1. Note that this padding only occurs in the CRC calculation, but is not part of the sent message. Once given the CRC result, we look it up in the table. If the result is not present in the table or the indices of the supposed errors are greater than the length of M, we know that we cannot correct the data. Otherwise, we correct the errors indicated by the table.

This method is undesirable for memory-constrained situations because it requires a large table with an entry for every possible error sequence. However, this method is still useful for determining polynomial validity.

3.2 Cyclic Method

3.2.1 *Theoretical Justification*

We can exploit the following property of cyclic codes (shown in [18]) to improve the memory requirement. If $s(x)$ is a CRC polynomial, $g(x)$ is the generator polynomial, and $m(x)$ is the message polynomial, then by the definition of binary polynomial division we know $m(x) = q(x)g(x) + s(x)$. Let k be the length (aka degree) of $m(x)$. Then it can be shown [18] that if one cyclically shifts $m(x)$ i times (modulo $x^k - 1$) to compute $m'(x)$, the new remainder (aka CRC) of $m'(x)$ when divided by $g(x)$ will be equal to the cyclically shifting $s(x)$ i times modulo $g(x)$. To shift $m(x)$ modulo $x^k - 1$ is a simple operation: If $m(x) = a + bx + cx^2 + dx^3$, then $m'(x) = d + ax + bx^2 + cx^3$. Or in binary, a wrapping right shift.

To cyclically shift the CRC $s(x)$ is a bit more complicated: one must multiply by x, and then take the remainder of $xs(x)$ when divided by $g(x)$. However, this is equivalent to the procedure for computing the CRC of a stream of data when we have a zero bit as input [18]: we shift the input register (a multiplication by x) and add $g(x)$ if the rightmost bit was one (division by $g(x)$). Since CRCs are a cyclic code, this means that we can perform a wrapping right shift of the message and, instead of recalculating a new CRC, simply cyclically shift it one iteration.

From section 3.1.1, we know that we can take an error sequence and match the CRC with the message if the errors are in the same place. However, now we can extend this: If the errors are in any cyclically shifted location, all it takes is a few cyclic shifts of the message CRC and it will match with the CRC of the error sequence. Also, we only need to shift the CRC and check at most L times, where L is the length of M. Assuming we place the error sequences in the table shifted as far right as possible, it is easy to recover the bit locations: If we have shifted i times, the offset is i, since the errored bits have not had a chance to "wrap" yet.

[1]Sometimes i itself is used for this purpose.

Finally, we must address padding. In the tabular method the transmitter had to pad the data when computing the CRC, but the cyclic method does not require it. If the transmitter had padded the data, we can assume that they had padded with rightmost zero bits, which would be equivalent to cyclically shifting the CRC. So all that is required is for the receiver to cyclically shift the CRC extra times, shifting the index past the invisible padding bits and onto the actual data.

3.2.2 *Description*

In order to apply this method, the error pattern must be formulated cyclically. This means declaring a set of subpatterns, that we will call kernels[2] which have fixed length, and a stride. Then the pattern contains all sequences formed by shifting that kernel through the data at a specified stride. For example, if we have kernels (101,111) and stride 2 then the error sequences will look like {1010000, 1110000, 0010100, 0011100, . . . } . Note that all patterns can be expressed in this format by placing each sequence in the kernel set. However, a small number of kernels is desirable to conserve space.

Similar to the tabular method, all kernels are stored in a table. Then, when we have a message we want to correct (with a nonzero CRC) of length L, a maximum message length X, and a stride S, we perform $X - L$ different cyclic shifts of the CRC with result R. As before, a cyclic shift is the same as a bit-by-bit CRC computation with input bit 0. Then we compare the current R with the kernel set. If R maps to pattern P, we know P occurred in the data at offset 0. If not, we perform S more shifts and lookup the CRC again, and if it matches we know the offset is S and can correct the data. We continue this process until the offset is greater than the data length, at which point we know the error is not correctable.

This method uses much less memory, for example it uses 64 bytes to correct 4-bit-bursts in 40 bytes of data, while the tabular method uses 10KB. This improvement comes at the cost of extra computation equivalent to computing a single CRC of packet length X using the bit-by-bit method. It is possible to optimize further, see section 8.2.2.

3.3 Maximum Message Size

The choice of X greatly affects the performance of both algorithms. Too high of a limit causes too many additional error sequences, lowering the chances of creating a collision-free table, which in turn limits available generator polynomials and correctable error patterns. Also, a high message size can increase the amount of computation needed by the error correction algorithm, whether through additional padding in the tabular method or additional shifts in the cyclic method.

4. OPTIMIZING CRC ERROR CORRECTION FOR A GIVEN ENVIRONMENT

In general, the first step to deploying an error correction system is to determine what type of errors are induced by the hardware and the environment. To do this, we performed

[2]These are traditionally called generators, but we wanted to avoid confusion with "generator polynomials".

a thorough analysis of an error trace containing data transmitted wirelessly by MicaZ motes in a noisy environment (these traces are described fully in section 6, where they are used in our simulations). Our findings indicate that single 4-bit bursts aligned at the half-byte boundary (which we call half-octet errors) are the most common type of error, accounting for at least 50%[3] of all errors in our trace.

The reason for this is due to the MicaZ's radio, the Chipcon CC2420, and how it translates chip sequences to bits. According to [22], every 4 bits of data are mapped by radio hardware to a 32-bit chip sequence before they are transmitted, and back again when they are received. If the chip sequence does not match one in the mapping, the closest chip sequence is used. Thus, if a burst error occurs in transmission, it is highly likely to be contained in a single 32-bit chip sequence, and map to a half-octet error, regardless of the original length of the burst. If there is only a small amount of noise (e.g. arising from 802.11 interference) in the environment, the hardware will usually be able to decode the correct 4-bit sequence, but sometimes it will not, resulting in occasional half-octet errors.

Although the traces we used to identify these errors were taken from the MicaZ, since we have traced the cause back to the hardware, the same errors should be common in other devices using the CC2420, such as the Crossbow TelosB, Sun SPOT, and Intel iMote2[4]. Also, similar radios such as the CC2520 have the same symbol-to-chip mapping and thus also tend towards half-octets [23]. Finally, even on very different hardware one is likely to find well-defined and peculiar error patterns. For example, in [3] researchers observe that the pattern 10010001 occurs quite frequently due to an artifact of their radio hardware. Whether the identified pattern is half-octets, short bursts, or 10010001, identifying these patterns leads to information theoretic gains. This is because if the entropy of the possible error sequences is low, one should need fewer bits of redundancy to correct them, no matter the length of the error. However, schemes such as Reed-Solomon lack the ability to adapt to any arbitrary error pattern, because they aim to correct all errors of a given byte length. Our scheme, on the other hand, is much more flexible and can easily adapt to the exact error pattern. Because of this, we were able to focus CRC error correction on correcting half-octet errors in our simulations and experiments. Since the standard CRC-CCITT generator polynomial is not valid for error correction beyond 5-bit-bursts, we restrict ourselves to single half-octet errors. Focusing on this small class of errors allows us to decrease our probability of miscorrecting the data while correcting a high percentage of the observed errors.

5. TVA

Reliability is critical for most networks, and hence CRC error correction seems dangerous: Since we use some of the entropy in the CRC to correct a class of errors (such as half-octet errors), according to information theory we cannot retain the same error detection power. So if, after running the error correction procedure, we simply pass the resulting packet up the stack, we risk passing a packet that is still corrupted. Since we can no longer use the CRC itself to verify that the packet is correct (as explained in section 3.1.1), the only way to be certain the message is indeed correct is to contact the sender.

So in TVA, after correcting a message, the receiver sends a verification message to the sender containing a description of the original packet. The sender checks the description against its records and responds with either a resend if the description is incorrect, or a very short confirmation message if the packet is correct. The choice of packet description for the verification message is critical: it needs to be very short and easy to compute, yet it needs to provide very high confidence that if the description matches, the corrupted packet was successfully corrected.

TVA uses a 16-bit CRC for this message description because it is highly efficient to compute and can detect many errors with a short code. However, now one must choose the best generator polynomial to use, as a poor generator polynomial will result in poor error detection ability and thus poor reliability. Re-using the standard 802.15.4 polynomial is not an option, since we have already used it for error correction and thus it will always fail to detect mistakes in the correction procedure, as explained in section 3.1.1. Instead, we must specify the desired properties of the new polynomial, and then efficiently search the space of polynomials to find the one with the best error detection properties.

5.1 Generator Polynomial Properties

The first observation is that since it is not necessary to protect the integrity of the original CRC, we can exclude it from the checksum calculation.

Label the original generator polynomial G, and the potential generator polynomial P. A message M is transmitted with CRC r_0, and arrives at the receiver corrupted with error pattern e_1. The receiver corrects this message by XORing in error pattern e_2, based on the table mapping. We assume $e_1 \neq e_2$, otherwise the correction has succeeded. By the linear properties of CRCs explained in section 3.1.1, we know the new CRC of this corrected message is zero. Thus where q_0 is a quotient,

$$(x^n M \oplus r_0) \oplus e_1 \oplus e_2 = q_0 G \qquad (1)$$

In order for TVA to fail, the CRC computed over the second generator polynomial must match. Let e_i' be the quotient of $\frac{e_i}{x^n}$. So:

$$CRC_P(M \oplus e_1' \oplus e_2') = CRC_P(M) \qquad (2)$$

Since the first term represents the corrupted message with the last n bits removed. By the linearity of CRCs:

$$CRC_P(M) \oplus CRC_P(e_1' \oplus e_2') = CRC_P(M) \qquad (3)$$

$$CRC_P(e_1' \oplus e_2') = 0 \qquad (4)$$

In other words, the key property of the polynomial P is that it does not fail to detect a message error consisting of a common channel error XOR'd with a correction error (in our case a single half-octet), in the cases for which the original CRC G detects but does not correct the original channel error pattern. Note that the optimal P is probably not a standard polynomial, since standard polynomials were created to detect burst errors, not this special class of error pattern.

[3]The rest of byte errors accounted for about 20%, 2-byte errors for 15%, 3-byte errors for 5%.

[4]In fact, together with our colleague Mark Grossman we have verified this in the case of the Java-programmable Sun SPOT.

5.2 Searching for Polynomials

Now that we have defined the desired properties (we call a polynomial satisfying these properties for some channel pattern *TVA-valid*), we still have to find the generator polynomials that satisfy these properties over the most inclusive set of channel errors. While simple properties of polynomials can be shown mathematically, more complicated properties must be evaluated empirically [11]. To do so, one must define what error patterns one wishes to guarantee that they will avoid. Because from section 4 we know errors occur in half-octets, we consider two error patterns: k-half-octets and k-half-octet bursts. We are interested in TVA-validity for maximum message sizes of 41 (TinyOS's maximum message size) up to 127 (802.15.4's maximum message size).

Searching through all polynomials for a given message size is computationally intensive, and can take up to a week for a single message size on commodity hardware. As such, we would like to show that polynomials which are not TVA-valid at a smaller size also are not TVA-valid at all larger sizes, given the same channel error pattern. If we show this, then if a polynomial satisfies our criteria at a given size, we know it also does at smaller message sizes. Similarly, if a polynomial is not TVA-valid at a given message size, we do not have to retest it at larger sizes.

5.2.1 *Proof of monotonically diminishing TVA-valid polynomial sets*

To prove this property, we need to show that if P causes TVA failures for a message M, then it causes TVA failures for xM as well. First we show that the original CRC still detects the error (so correction can proceed), meaning CRC_G of the corrupted xM is not 0.

We will actually show something stronger, so that we may reuse the proof. If $CRC_G(m_A) \neq CRC_G(m_B)$, then $CRC_G(xm_A) \neq CRC_G(xm_B)$. If m_A is taken to be the corrupted message $(x^n M \oplus r_0) \oplus e_1$, and m_B is taken to be 0, then what we want to prove becomes a special case of this more general property.

So, we will show that if and only if G is an odd polynomial,

$$CRC_G(m_A) \neq CRC_G(m_B) \implies CRC_G(xm_A) \neq CRC_G(xm_B) \quad (5)$$

Since the standard CRC-16-CCITT (and most other commonly used generator polynomials) are odd (x^0 coefficient is 1), this will demonstrate the required property. So we begin with the forward direction, assuming G is odd:

$$m_A = q_A G + r_A, m_B = q_B G + r_B, r_A \neq r_B \quad (6)$$

$$\frac{m_A}{G} = q_A + \frac{r_A}{G}, \frac{m_B}{G} = q_B + \frac{r_B}{G} \quad (7)$$

So, for a contradiction, the remainder of $\frac{r_A}{G}$, r'_A must equal the remainder of $\frac{r_B}{G}$, r'_B. We split into three cases based on the degree of r_A and r_B. It should be clear that since r_A and r_B are remainders from a division with G, their degree must be less than the degree of G.

Case 1:

$$deg(r_A) < deg(G) - 1, \quad deg(r_B) < deg(G) - 1 \text{ so,} \quad (8)$$

$$deg(xr_A) < deg(G), \quad deg(xr_B) < deg(G) \quad (9)$$

$$\frac{xr_A}{G} = xr_A = r'_A, \quad \frac{xr_B}{G} = xr_B = r'_B \quad (10)$$

But $r_A \neq r_B$, so $xr_A \neq xr_B$ and thus $r'_A \neq r'_B$, a contradiction.

Case 2:

$$deg(r_A) = deg(G) - 1, \quad deg(r_B) = deg(G) - 1 \quad (11)$$

$$\text{In this case, } deg(xr_A) = deg(G), \quad deg(xr_B) = deg(G) \quad (12)$$

$$\text{So } xr_A = G + r'_A, \quad xr_B = G + r'_B \quad (13)$$

But for a contradiction we assumed $r'_A = r'_B$, so by equation (13) $xr_B = xr_A$, so $r_B = r_A$, a contradiction.

Case 3: Without loss of generality,

$$deg(r_A) < deg(G) - 1, \quad deg(r_B) = deg(G) - 1 \quad (14)$$

$$\text{So } xr_B = G + r'_B \quad (15)$$

$$xr_B - G = r'_B \quad (16)$$

$$\frac{xr_A}{G} = xr_A = r'_A \quad (17)$$

By our assumption that $r'_A = r'_B$:

$$xr_A = xr_B - G \quad (18)$$

If we examine the constant (x^0) term on the left side of the equation, it is zero. The only way for it to be zero on the right side of the equation is if G has a zero constant term. But G is odd, a contradiction. □

Now for the backward direction of (5): No even polynomial G can satisfy the property: If $CRC_G(m_A) \neq CRC_G(m_B)$, then $CRC_G(xm_A) \neq CRC_G(xm_B)$ for all messages m_A, m_B.

Let

$$m_A = \tfrac{G}{x}, m_B = 0 \quad (19)$$

Since G is even, $\frac{G}{x}$ is a polynomial without remainder, so m_A is a valid bit sequence. $CRC_G(m_B) = 0$, and since $deg(\frac{G}{x}) < deg(G)$, $CRC_G(m_A) = \frac{G}{x}$, so m_A and m_B satisfy the antecedent.

$xm_B = 0$, so $CRC_G(xm_B) = 0$. $xm_A = x\frac{G}{x} = G$, and thus $CRC_G(xm_A) = 0 = CRC_G(xm_B)$. □

Next we must show that the correction will proceed with the payload xM, namely that it will map $CRC_G(xe_1) = CRC_G(xe_2)$. Since the table required $CRC_G(e_1) = CRC_G(e_2)$, it suffices to show a more general property, that:

$$\text{For all } G\text{, If } CRC_G(m_A) = c = CRC_G(m_B)\text{, then } CRC_G(xm_A) = CRC_G(xm_B) \quad (20)$$

$$m_A = q_A G + c, \quad m_B = q_B G + c \quad (21)$$

$$xm_A = xq_A G + xc, \quad m_B = xq_B G + xc \quad (22)$$

$$\frac{xm_A}{G} = xq_A + \frac{xc}{G}, \quad \frac{xm_B}{G} = xq_B + \frac{xc}{G} \quad (23)$$

Hence the remainder of both messages is the remainder of $\frac{xc}{G}$, so they are equivalent. □

So the correction table will still map xe_1 to xe_2. Finally, we want to show that the CRC taken with P will still fail to detect the error (refer to (4)), or:

$$CRC_P\left(x\left(e'_1 \oplus e'_2\right)\right) = 0 \quad (24)$$

This follows from the property shown in (20), letting $m_A = e'_1 \oplus e'_2$, and $m_B = 0$. □

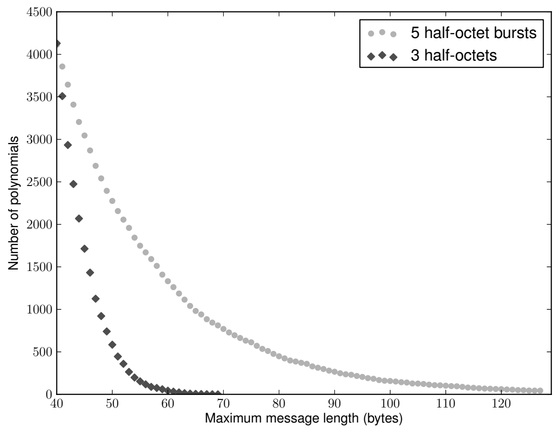

Figure 1: Polynomials that cause no TVA failures on up to five bursts of half-octet errors (light line) or no TVA failures on up to three half-octet errors (darker line).

5.2.2 *Retesting error sequences*

Another optimization question is, must we retest all error sequences for each new message size, or, if no TVA errors were found in a shorter message size s, can we ignore those sequences in which the errored bits only occur within s? We reason through this property below:

We need to show if any message M does not have a CRC_P collision in the final step, xM will also have no collision. We can apply the properties shown in (20) and (5) to show that for standard G, xM passes the CRC check if and only if the check succeeded for M. Then, we apply our proof of (5) and (20) to show that for standard G, xM will cause a table mapping to xe_2 if and only if M caused a table mapping to e_2. Now we wish to claim that, assuming $CRC_P(e_1' \oplus e_2') \neq 0, CRC_P(x(e_1' \oplus e_2')) \neq 0$. But (19) shows that if $e_1' \oplus e_2' = \frac{G}{x}$, this does not hold for even P. Hence, if we want to consider even choices of P, using this optimization is not advised.

5.2.3 *Polynomial Search Results*

We found a large percentage of TVA-valid polynomials at 127 bytes for 2 half-octets and 4 half-octet bursts, but none even at 40 bytes for 4 half-octets or 6 half-octet bursts. Hence we set out to map out the space of TVA-valid polynomials at 3 half-octets and 5 half-octet bursts.

Using the first optimization and parallelizing the computation over several cores, we were able to search through the space and create the graphs shown in Figure 1. The results show that at 41 bytes, many polynomials have no failures for all three half-octet errors anywhere in the data, and many that have no failures for bursts of up to 5 half-octets[5]. There are no more of the former type at message sizes greater than 68, but a few of the latter type remain up to 127 bytes.

5.3 Evaluating TVA on a channel model

This search allows us to find the polynomials with the strongest TVA detection power. But it remains to be shown that the detection power of even the best polynomials allows TVA to reach the standard of reliability set by ARQ. Due to

[5]We require no failures for 2 half-octet errors as well.

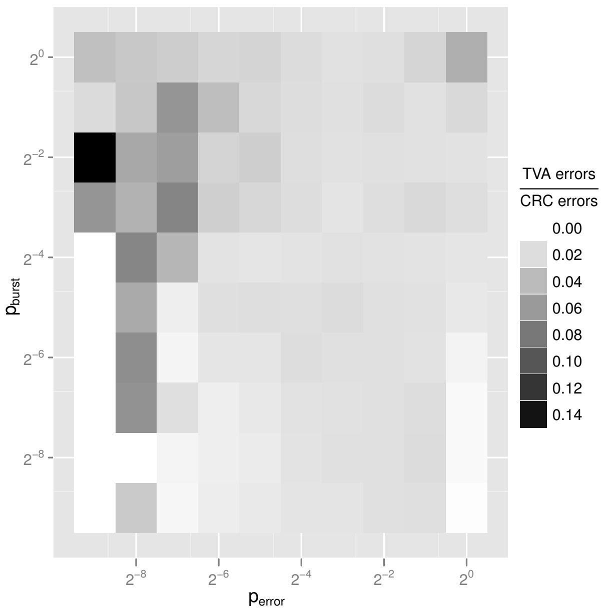

Figure 2: Relative reliability of TVA compared to CRC on 41-byte messages, using the TVA polynomial Q, and evaluated across channel parameters. For each setting of the parameters, we simulated TVA over 5×10^8 messages.

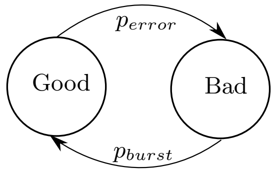

Figure 3: The Gilbert Elliot channel model. This is used only for determining the reliability of TVA in different channel conditions.

the large quantity of data needed for ARQ to fail to detect an error, evaluating this property in error traces or experiments is not feasible. As such, we evaluate this property using the Gilbert-Elliott channel model [4], which is a very popular two-state Markov model (Figure 3). However, this model does not accurately capture the tendency toward half-octet errors. As such, we modify the model so that the bad state simply outputs a random half-octet, which is a reasonable approximation of what occurs on CC2420 hardware when there are several errored bits in the chip sequence.

We vary the transition probabilities p_{error} and p_{burst} to inspire confidence in the reliability of TVA across wireless environments. In Figures 2 and 4 we present a few simulation results, using the

$$Q = x^{16}+x^{15}+x^{14}+x^{11}+x^{6}+x^{5}+x^{4}+x^{3}+x^{2}+x+1 \quad (25)$$

polynomial (0xC87F in hex), which we found to have exceptional reliability guarantees[6], able to detect all TVA mis-

[6]To pick this polynomial, we used the data displayed in Figure 1 to determine the maximum data length where there were remaining polynomials that could detect all 3 half-octets and 5 half-octet-bursts. That maximum length was 63, with only two polynomials, one of which was 0xC87F.

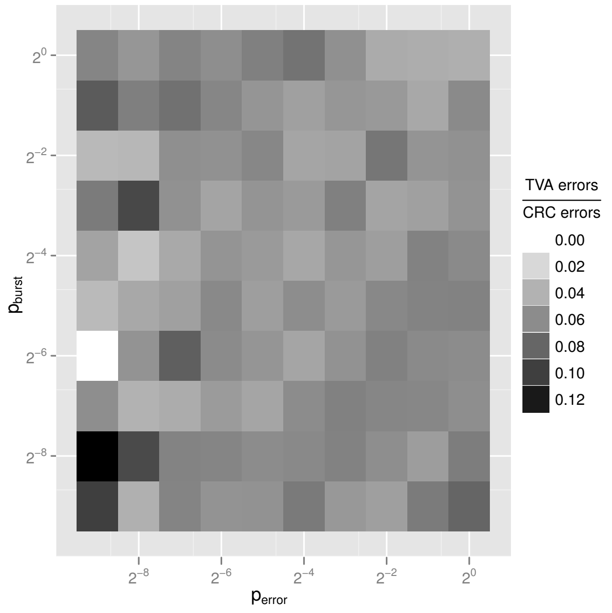

Figure 4: Relative reliability of TVA compared to CRC on 127-byte messages, simulated over 1×10^8 messages.

takes with 3 half-octet errors on messages of length up to 63, and up to 5 half-octet bursts on messages of length up to 70.

As shown in the heatmaps (Figures 2 and 4), the error rate is much lower (roughly 1/10) of that of the original CRC failing. While this obviously does not mean TVA has a lower probability of failure than ARQ (the CRC can fail in TVA as well), it means that TVA has a negligible increase in the error rate compared to ARQ.

6. SIMULATION SETUP

6.1 Traces Used

To evaluate the performance and demonstrate the versatility of our scheme, we decided on a trace-driven evaluation methodology, using WSN traces collected by others. We found a suitable suite of traces on the CRAWDAD (Community Resource for Archiving Wireless Data At Dartmouth) database. The trace was collected by Adnan Iqbal, Khurram Shahzad and colleagues [7] at the NUST school of Electrical Engineering and Computer Science, Rawapalindi. Iqbal et al. selected the Crossbow MicaZ sensor motes running TinyOS and using the 802.15.4 protocol. They placed a single sender mote at various locations while keeping a base station in a fixed position. The sender mote transmitted 31-byte frames, which include 11 header bytes and 20 data bytes, at a rate of 10 frames per second. Iqbal et al. collected 4 trace sets representing 4 highly diverse locations (PhD Lab, Outer Room, etc.) for the sender mote, holding the base station in a fixed position. Each set contains 6-8 different trials, each containing an average of 31,000 frames.

6.2 Simulation details

The trace data had some limitations. First, it contained message headers, some of which were corrupted, but there was no way to tell what the correct values of those headers should be. Second, there was no CRC recorded. To overcome these limitations and to allow us to corrupt additional bytes sent, we transformed the recorded packets into a steam of bit errors. To do this, we recorded the index of each corrupted bit in the payload. Then, when we simulate sending a message, we XOR the next bits of the stream with the message, thus replaying the errors. We constructed a packet of 31 bytes with an 11-byte header and a fixed payload, similar to the original data. We also generated different sequence numbers to use in the header. To simulate this fairly, each protocol sees the same errors in the initial data packet. We evaluated these metrics over two popular packet sizes: TinyOS's maximum packet size of 41 bytes and 802.15.4's maximum packet size of 127 bytes.

We measure the simulated average code rate (defined as data bytes successfully received over the total bytes sent), which is a more theoretical metric, and simulated throughput and energy consumption, which are more practical measures. With regard to energy consumption, reports such as [24] suggest that transmission and reception of data are both power-intensive operations, but each take about the same power on hardware similar to the CC2420. So this indicates that the predominant factor affecting energy consumption is how long the process of packet transmission, reception, and acknowledgment takes before the radio can be turned off. Thus, to maximize both throughput and energy efficiency the goal is to determine how much time each protocol takes, given the same amount of data to transmit.

Estimates for our timing simulation are taken from our investigations into TinyOS. In our simulation, every protocol takes a very short round trip time ($\xi = 10$ ms) to successfully send a message and receive a response, but in the case of a retransmission the waiting time is longer ($\gamma = 100$ ms), since the link layers try to backoff to overcome the perceived noise.

We simulate TVA using the polynomial specified in equation 25 and target it to correct single half-octet errors (section 4). While the implementation uses a complex buffering scheme (see section 8.2.4), we simplified this slightly for the purpose of the simulation. The sender takes only RTT to respond with an TVA-ACK message, so we estimate the time of a successful TVA exchange at 2ξ. If the TVA-2 message is corrupted, we wait for a relatively short period of time before resending, about 50 ms, in order to attempt to contact the transmitter before a message is resent. While in our actual implementation (see section 8.2.4) the receiver gives up sending TVA-2 eventually and can accept a response to any message, here we are pessimistic and assume the sender can only respond to the first two resent TVA-2 messages, but the receiver keeps sending TVA-2 messages afterwards. Lastly, we do not simulate the limit on TVA-2 buffers, because we found this limit was hit rarely in practice.

Finally, since our goal was simply to get a record of data sent, we saw no need to fully simulate the HARQ protocol. We simply calculated the number of additional bytes required for every packet and counted the number of packets that had uncorrectable errors (based on the theoretical error correction properties, see section 2.2).

7. SIMULATION RESULTS

7.1 Analysis of Simulation Results

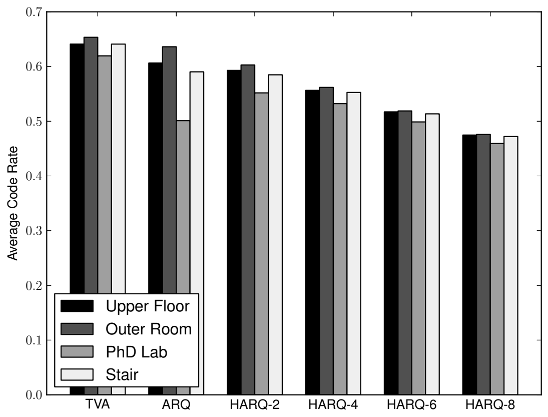

Figure 5: Average code rate with 41 bytes of data.

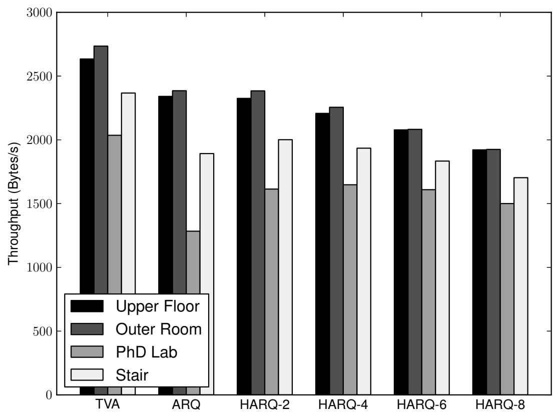

Figure 6: Throughput for 41 bytes of data.

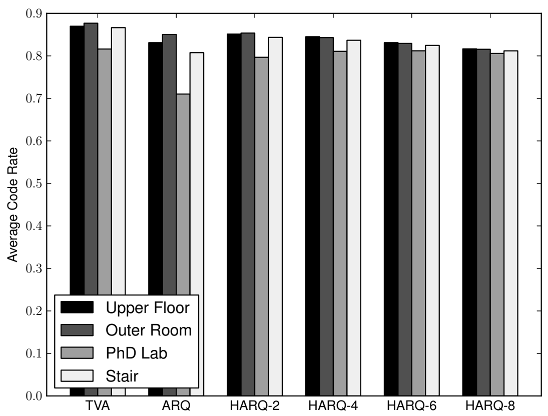

Figure 7: Average code rate with 127 bytes of data.

Figures 5-8 show the simulation results, averaged over all the traces in a given environment. These are averaged over 6-8 trials in each environment, and due to small changes in the configuration between trials, the PER can vary significantly across trials even in a single environment. As such, below we attempt to give a better understanding of the variance in results across individual trace sets.

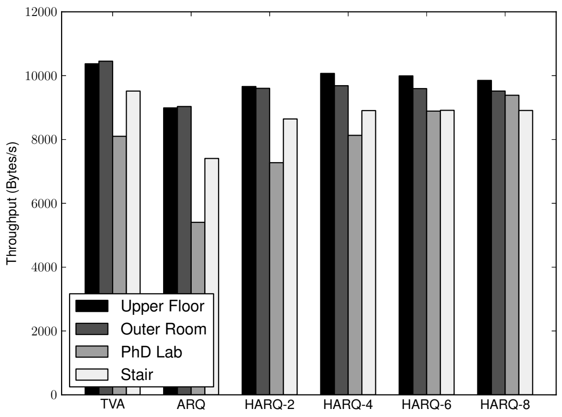

Figure 8: Throughput for 127 bytes of data.

7.1.1 Error Correction Reliability

TVA, ARQ and HARQ pass no errored packets to the application layer in all of our simulations. To get some sense of how frequently residual errors would be expected if we did not do the TVA confirmation, we also tested a protocol which assumed that if CRC error correction succeeded then the message was correct. We found that this protocol caused correction failures 0.36% of the time in error cases at 41 bytes, and 1.04% at 127 bytes. In most systems even a 0.36% times PER corrupt packet rate is unacceptable, justifying the reliability component of TVA.

7.1.2 Performance Comparison

TVA always performs better, both in terms of average code rate and in terms of throughput, than the commonly used ARQ protocol, no matter the message length. At 41 bytes, TVA does on average 62% better in throughput than ARQ, in some high-error traces more than tripling its throughput. Even at 127 bytes it does better in throughput by an average of 21% and a maximum of 68%. In fact, TVA only has the potential to perform worse than ARQ if we hypothesize a large number of correction failures, which would require sending both TVA verification messages and resends. However, as we have observed in 7.1.1, the empirically-estimated probability of these failures is only 1%, so this helps explain TVA's dominance over ARQ.

TVA also fares well against HARQ, almost always performing better in the 41-byte case (except in 12% of cases, which have very high PER), improving by up to 38% in throughput, and 17% in average code rate. In the 127-byte case, we still see a maximum throughput improvement over HARQ of 28%, although in about 25% of high error (>20% PER) rate cases TVA does worse than HARQ in throughput. This is due to the fact that with extremely long messages and high error rate, TVA cannot correct enough errors to beat the performance of a more powerful coding scheme. As shown in the summary graphs (Figures 5-8), TVA shows strong performance in both throughput and code rate when averaging over a variety of trials in the same environment.

Data	CRC	RS code

Figure 9: The format for HARQ message. The CRC acts as an inner code, while the Reed Solomon code acts as an outer code.

8. IMPLEMENTATION

A trace-driven simulation, while useful, is only one component of a rigorous evaluation of our system. An implementation on actual sensor hardware is required to fully demonstrate the effectiveness of our scheme. We chose to implement our protocols on Crossbow MicaZ motes running TinyOS version 2.1.0. We chose this platform because of its popularity in the WSN community. These devices are highly memory constrained, containing only 4K of RAM and 128K of ROM, and possess a 8MHz AVR ATmega128 processor. They are equipped with the CC2420 radio, which is designed to be compatible with the IEEE 802.15.4 standard. This radio is used by many other popular sensor motes, from the Crossbow TelosB to the Intel iMote2 to the Sun SPOT.

8.1 HARQ

In order to ensure that our implementation of Reed-Solomon was as efficient as possible, we use the TinyRS library, a Reed Solomon library optimized for sensor motes, developed by Liang et. al. [12]. However, the version of the library generously provided by Liang used so much RAM that it could not be installed on the MicaZ motes (see Figure 12). We thus further optimized[7] TinyRS to reduce its RAM consumption, and used the optimized version in our experiments.

HARQ type I operates similarly to ARQ, however the packet format is quite different. As seen in Figure 9, the CRC still is computed over the payload and attached to its end, as an inner code. But the FEC code is then used as an outer code, which wraps both the payload and the CRC. When a packet is received, the CRC is checked first, which avoids the high decoding time in the case of an uncorrupted packet. If the CRC check fails, the decoding process is done, at which point the CRC is checked again to ensure that the decoding completed successfully.

8.2 TVA

We implement a version of TVA that uses the polynomial specified in equation 25 and targeted to correct single half-octet errors (section 4). Here we describe some challenges implementing TVA on actual resource-constrained motes.

8.2.1 *Hardware Calculated CRCs*

In theory, CRCs are calculated as described in section 2.1. However, modern hardware such as the CC2420 does not compute CRCs via standard binary polynomial division. It turns out that when implementing the algorithm in hardware, the bytes of the data are reversed before the CRC is taken. In other words, a byte with bits 01234567 turns into 76543210.

This difference cannot be ignored as an implementation detail because it changes the way error patterns are defined. For example, the error sequence 0000001111000000 crosses a byte boundary, and so is equivalent to an error of 1100000000000011 in the transmitted data. Since the CRCs of these messages are different, the modification has a potential to cause a collision with other entries in the table, leading to differences in polynomial validity for error correction. For example, the standard 802.15.4 CRC-16-CCITT polynomial is valid, using the hardware calculation, for 4-bit-bursts over 48-byte messages, while using the standard calculation it is valid for over 128 message bytes. Fortunately, half-octet errors do not have this issue, as they do not cross byte boundaries, and thus reversing the bytes of a half-octet error yields another half-octet error. Hence, this method does not affect any theoretical CRC properties for half-octet errors, though it may for other error patterns. For backwards-compatibility reasons, TVA uses the same CRC calculation as the radio hardware, but since we focus on correcting single half-octet errors, our approach is still valid for over 128 message bytes.

8.2.2 *Efficient Table Lookup*

While in section 3.2 we showed how a CRC correction table can be easily optimized to save space, a naïve implementation of the table would increase compute time significantly, because, for the case of half-octet errors, all entries of the table must be considered after every 4 bits of data are consumed. Here we discuss how to optimize the table to fit memory requirements of the motes, namely that RAM is much more expensive than ROM, while attempting to reduce the computation time as much as possible.

The first optimization step is to make the correction table compatible with the byte-by-byte efficient CRC computation [18], which can be done by considering the stride to be 8 bits instead of 4, which doubles the size of our table from 15 to 31 CRCs. However, placing this table into RAM (for quick access) would take too much memory. Instead we must put it in ROM, but searching through all 31 entries for each byte of the received message takes significant computation time. To optimize further for time, we consider a two-tiered table approach: The first tier is indexed by the low-order byte of the CRC, it has 256 entries each one byte long. If the low-order byte is not in the original 31-entry table, it has a zero in this table, otherwise the first entry contains a index into the second table. The second table contains 31 2-byte values, containing the second byte of the CRC and the error pattern for that byte. Hence, one checks that the high-order bytes match, and if so, XORs the error pattern into the data to correct the error. In summary, we use only slightly more ROM for much better time complexity.

8.2.3 *Message Formats*

Since TVA requires extra packets to be transmitted over the network, it requires some instrumentation to implement on top of the existing functionality of the network stack. We had to additionally create two new packet types for the TVA-2 (verify) and TVA-ACK message. To differentiate them from normal packets, we chose frame types that were unused by 802.15.4, see Figure 10. Additionally, we set a reserved bit in the header of a data transmission to indicate to the receiver that the sender is TVA-capable.

8.2.4 *Buffering*

While the general description of the operation of TVA in section 5 seems simple enough, implementation of such a scheme requires additional complexity at the link layer, primarily in the form of buffering. While in a traditional

[7]Optimizations included moving data from RAM to ROM and reducing the data width of certain variables.

FCF(type:4)	DSN	TSN	TVA-CRC	FCS

FCF(type:5)	TSN	FCS

Figure 10: The format for TVA-2 and TVA-ACK messages, following the notation of [22]. The TVA-2 message is 8 bytes long, and contains the TVA CRC, and a TVA-DSN (TSN) to identify the TVA exchange. TVA-ACK is a 5-byte message, and follows the format of a traditional ACK, the only difference being the frame type in the Frame Control Field (FCF). A Frame Check Sequence (FCS) containing a CRC is appended for error detection only.

network setting, moderate buffering is not an issue, on the RAM-constrained motes buffering is very costly. Hence, any protocol proposed must have tight limits over the number of buffers required at the transmitter and receiver in order to be practical in a sensor network setting. Here we present a buffering implementation which requires a relatively small amount of RAM without loss of reliability.

The main reason buffering is needed is that if the receiver receives a message which it can correct, it must hold this packet at the link layer while it sends the TVA-2 message, and wait until it receives a TVA-ACK message or a resend. So, in the worst case, the mote must remember one packet for each neighbor it is communicating with. The problem becomes more severe if one considers that if the channel becomes briefly noisy, it may increase throughput if the mote can transmit a new TVA-2 message for multiple corruptions of the original message. Additionally, the mote must periodically resend the last transmitted TVA-2 message, because if the transmitter successfully received the message and their TVA-ACK was lost, the sender needs prompting to resend the TVA-ACK. Hence, what appears to be a simple protocol can be surprisingly complex to implement.

Our buffering scheme is as follows: The receiver has a fixed set of u buffers, which may be used for holding received messages. When it receives a corrupted message, it checks the buffers to determine if one is free before trying to correct it. If the correction succeeds, the message is copied[8] into that buffer. All other buffers with the same Data Sequence Number (DSN) as the received message are now marked as unsendable. Periodically a timer fires on the mote instructing it to resend any outstanding TVA-2 messages[9]. If one is found with no remaining resends, it and all other records with that DSN are deleted. Otherwise, that message is sent and its resend counter is decremented.

The default TinyOS link-layer transmission component makes the assumption that only one message may be sent at a time. This simplifies things slightly, because we do not have to remember the entire contents of messages we have transmitted. Instead, we can simply store short records containing the TVA-2 CRC, the DSN of the packet, whether the request is complete or not, and if so what was the correct TVA-2 DSN. Even though these buffers are very small, they also must be carefully managed, because if we simply create one for every new message there is no bound on the number of buffers needed. The penalty for forgetting a message too soon is large, as it means the application will believe that message was transmitted when in actuality it was not.

To overcome these problems, the transmitter stores at most 2 records for every destination address. When an ACK arrives, the corresponding record is deleted. When a correct TVA-2 comes in, the record is marked as completed. When a new message is ready to be resent, it replaces the oldest DSN[10]. To ensure that replacing the older DSN did not overwrite valuable information, we require the receiver to refrain from sending any more TVA-2 messages (ACKs are still allowed) if it is waiting for a response to previous TVA-2 messages with a different DSN from the same transmitter. Hence the buffer space for receiver and transmitter combined is

$$lu+2nc \tag{26}$$

Where l is the message length, u is the user-defined number of receiver message buffers, n is the number of neighbors, and c is a small constant describing the size of a TVA-ACK record (8 bytes in our implementation). This should be compact enough to be practical for most sensor motes. An important note is that if the number of neighbors is estimated too low, then the TVA transmitter can detect this by observing that there are too many destination addresses in its record list, and can send the message without its TVA-capable bit set (equivalent to performing ARQ).

8.3 Performance Comparison

Figure 12 shows that our implementation of TVA($l = 41$, $u = 3$, $n = 2$, $c = 8$) shows an impressive improvement in RAM over HARQ (52% less than than the unoptimized HARQ and 26% less than the optimized version), which is critical on the RAM-constrained MicaZ's. As for ROM, TVA and all implementations of HARQ use ~5k more than the default of ~20k. TVA and HARQ use essentially the same amount of ROM (within 4%), and ROM is not as critical of a commodity on the MicaZ, since it has 128k total. In terms of computation time, TVA is extremely fast compared to HARQ, since as shown in Figure 11 HARQ-2 takes up to $20\times$ the encoding time and $9\times$ the decoding time of TVA.

8.4 Backwards Efficiency

Backwards compatibility is important for many networking systems, and sensor networks are no exception. Sensors are often placed in difficult-to-access places such as radioactive plants or volcanoes, and if sensor software is to be upgraded it is much more efficient to do so piecemeal instead of bringing the entire system down and then back online. Even in networks where the sensors are easier to access, one may wish to evaluate a new protocol on a subset of motes before deploying it to the entire system. Lastly, if a protocol requires a large amount of energy or computation time, motes may desire the ability to conserve resources by switching to a simpler protocol at any time. While there are many notions of backwards compatibility in networks, with the weakest being that some meaningful communication can occur between devices running different protocols, a stronger notion is needed in order to ensure that communication can efficiently occur in such a mixed network. We argue that a protocol should be *backwards-efficient*, as defined below:

[8]Actually, no memory copying occurs because of its expense, rather the pointer to the buffer in which to place received messages is changed.

[9]Since one can only send a single message at a time, the mote processes TVA-2s to be resent in a round-robin fashion.

[10]TinyOS enforces strictly increasing DSNs, so this is easy to determine. In applications where this is not the case one could store a timestamp.

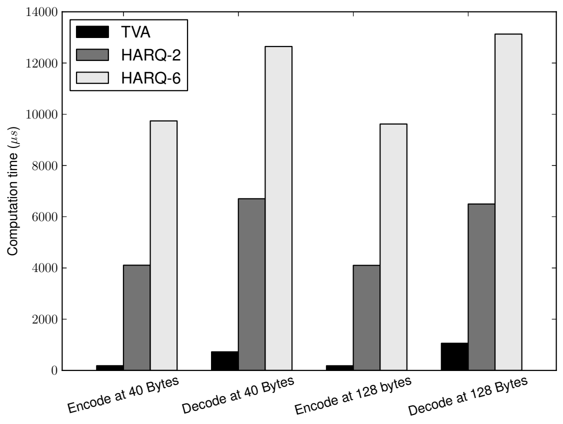

Figure 11: Timing measurements across libraries.

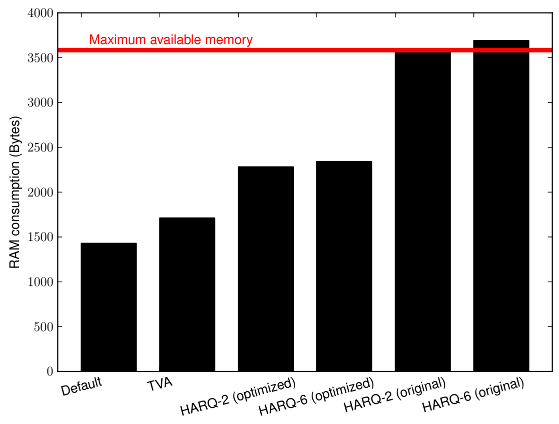

Figure 12: RAM consumption across libraries. The red line shows the maximum available RAM on the motes (of the 4K total ~512 bytes are needed for stack space).

- For protocol A to be backwards-efficient with protocol B, any number of nodes running protocol A must be able to communicate with any number of nodes running protocol B without any wasted transmission: That is, no bytes should be sent to a mote which are not useful to it due to it using the wrong protocol.

- For protocol A to be fully backwards-efficient with protocol B, any node running protocol A must be allowed to switch, for any new data transmission, to protocol B, and vice-versa, without sending any unnecessary data.

By this definition, TVA[11] is backwards efficient with 802.15.4. If the receiver is using 802.15.4 and the transmitter is using TVA for any message, no TVA-2's will be sent and thus it will appear as though both nodes are using 802.15.4. If the transmitter is using 802.15.4 and the receiver TVA for any message, the receiver will know not to send the TVA-2 message due to the fact the reserved bit in the header has not been transmitted in the data packet.

Note that no FEC scheme can be backwards efficient with 802.15.4. The reason is that by definition, at some point FEC must add additional redundancy to a data packet. When it does, a 802.15.4 receiver, even if capable of ignoring that redundancy, cannot use those transmitted bytes.

[11]Because our interaction with the radio causes a slight delay which causes it to be difficult to send an ACK to the transmitter in time, our current implementation is not fully backwards efficient with the default TinyOS stack. Simply increasing the ACK waiting constant from 7.8 ms to 12.2 ms will allow efficient interoperability, and because this number does not control when resends occur this modification should not have a sizable effect on throughput. Note that HARQ also requires this change.

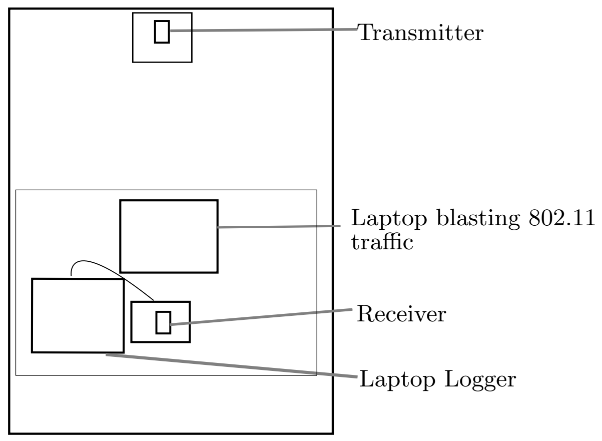

Figure 13: Experiment Setup. A laptop blasting WiFi traffic is placed between two motes. A logging laptop is placed to the side.

9. EXPERIMENTS

9.1 Experiment Setup

After implementing HARQ and TVA, we performed an experimental comparison of these two protocols to ARQ in order to demonstrate the real-world merit of our error correction scheme. In contrast to our simulations, which consider bit corruption resulting from distance and obstruction-based errors, our experiment is designed to demonstrate that TVA shows throughput and average code rate improvements in the presence of 802.11 interference. Since 802.11 operates in the same 2.4Ghz band as 802.15.4, interference from WiFi enabled devices has been widely recognized as a problem in sensor networks [12]. One solution to WiFi interference is to ensure proper separation of the channels used by both devices, however [12] argues that this is difficult in practice and improving coexistence of 802.15.4 and 802.11 on the same channel is often a more attractive solution. Although the source of the bit errors is WiFi interference rather than obstructions, we still consider correction of single half-octet errors, since as explained in section 4 the half-octet error pattern can be attributed mostly to the hardware. Instead of a complex multi-node testbed, we believe the best method to elucidate the differences in performance between protocols, is to use a simple, easy to analyze two node setup.

Our experiment used two MicaZ motes, one as a dedicated transmitter, the other as a dedicated receiver. The transmitter was placed at one end of a room on a small table.

The receiver was placed approximately 12' away on a table of equal height. The receiver was placed atop a MIB 510 programming board, which was connected via a serial-USB cable to a laptop. The laptop was placed 6" to the left of the mote, and its purpose was to record logging data sent through the serial cable. Approximately 3" in front of the mote we placed a second laptop, whose purpose was to generate the 802.11 interference. It ran the TTCP [20] tool in UDP mode, which broadcasted a continuous stream of 802.11 packets over the laptop's wireless interface. Though the laptop was in the line of sight of the motes, its placement alone did not contribute significantly to the error rate, as when we disabled TTCP the BER between the two motes dropped to 0.

One major obstacle we encountered when comparing several different protocols experimentally is that recreating the exact error conditions during multiple runs is exceedingly difficult[12]. Even making every effort to maintain the physical relationship between the equipment, the slight movements required to reprogram the motes between trials caused major differences in PER. Also, the channel quality varies over time, meaning even within a single run conditions were changing substantially. To attempt to overcome this, we used a two-pronged approach: a calibration phase before each trial, and multiple short trials for each protocol. For the calibration phase, we slightly modified the equipment positioning until the BER and PER were within a acceptable range. Then, we reset the receiver mote, and left the equipment unmoved for the remainder of the trial. We then ran each protocol multiple times in slightly different positions, with each trial being fairly short (1000 packets). This allowed us to more easily compare the performance of different protocols in a variety of error conditions. For a more direct comparison with less noise, see section 6.

9.2 Experimental Results

We evaluate the following 4 protocols experimentally: ARQ, TVA, HARQ-2 and HARQ-6. We created a simple application which sends unicast packets to a receiver using TinyOS' PacketLink functionality. Since there are no default values for the number of retransmissions and the time between retransmissions, we use the values suggested in TEP 127 [19]. All packet sizes are fixed to TinyOS's default maximum packet size. To simulate the fact that sensor network applications generally are concerned with periodic data collection, a minimum time of 50ms between packet transmissions at the application layer is enforced.

Each receiver logs summary statistics about number of bytes transmitted and received, PER and BER (before correction), and time elapsed since the first received packet. We did not consider it necessary to measure energy usage, since as explained in section 6.2 time and energy usage should be strongly correlated. These statistics are transmitted to the logging laptop every 100 successfully received packets.[13] We ran each protocol until the application had received 1,000 packets, and tested each protocol approximately 10 times.

The experimental results are presented in Figures 14–15 and Table 1. As expected, the result graphs show that for no errors (this data point was taken in a different environment without interference), results follow the expected trend: ARQ and TVA perform the best, while HARQ-2 and HARQ-6 perform worse due to their high overhead.

The graphs have some noise, but generally show TVA performing best, both in throughput and in average code rate, especially at lower error rates. This is partly due to the fact that TVA could correct 43% of errors without adding additional redundancy to the data packet. In comparison, HARQ-2 was able to correct 41% of errors averaged over all trials, compared to TVA's 43%, so HARQ-2 did not show a significant gain in correcting ability. While HARQ-6 was able to correct 74% of errors, this gain in error-correcting ability was not worth the 6 extra bytes of redundancy.

[12]Indeed, these differences have led others (e.g. [3]) to refrain from performing an experimental comparison altogether.

[13]Because of the high speed of the various protocols, we were unable to log more detailed information such as the exact error patterns encountered.

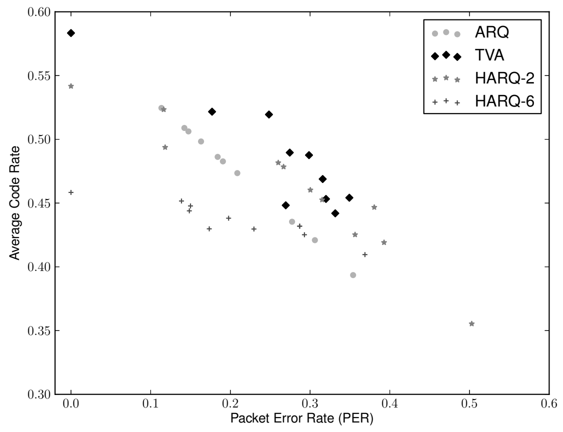

Figure 14: Experimental results for average code rate compared to PER.

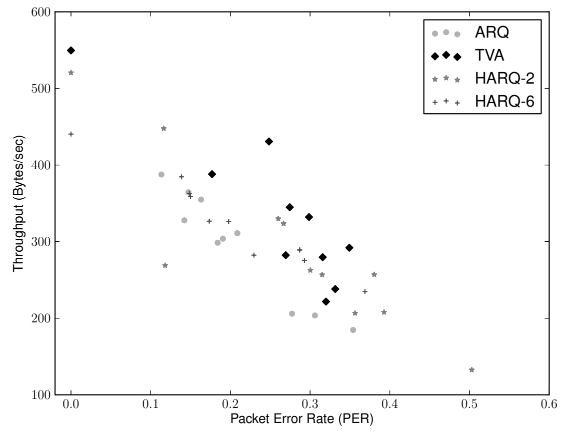

Figure 15: Experimental results for throughput compared to PER.

protocol	extra time per packet error (ms)	extra bytes per packet error
ARQ	172.72	42.00
TVA	**119.69**	**28.81**
HARQ-2	152.21	33.34
HARQ-6	131.69	54.84

Table 1: Summarized Experimental Results

To get better understanding of the experimental results, in Table 1 we present several summarized metrics. To compare across scenarios with different PERs, these metrics quantify how much extra time and bytes each protocol takes per data packet error (recall that all packets are the same length, making this a fair comparison). "Extra" refers to a comparison with the amount of bytes and time ARQ would take to transmit the same amount of data (calculated from a deployment of ARQ in an error-free environment).

Table 1 shows that TVA is more efficient in time and communication compared to ARQ and HARQ. Specifically, ARQ takes 42 bytes (the size of a resend, 41 bytes of the TinyOS packet protected by the CRC, plus one additional length byte) to recover from each data packet error (or encounter another errored data packet), while TVA is able to communicate less by correcting errors. We see that HARQ-2 does better than HARQ-6 in terms of bytes sent (since HARQ-6 adds extra additional redundancy to each data packet), but worse in terms of time (since HARQ-6 can correct more errors). TVA does better because it avoids this tradeoff, since additional redundancy does not have to be provided upfront. Overall, we see that TVA uses 31% less redundant communication and 30% less additional time to recover errored packets compared to ARQ.

10. CONCLUSION

In this paper, we present Transmit-Verify-Acknowledge (TVA), a link-layer error recovery protocol designed for low-error scenarios. TVA uses a novel method of CRC error correction to correct bit errors in resource-constrained networks, along with a carefully-designed confirmation exchange to ensure reliability. We show, through trace-driven simulations and sensor network experiments, that TVA improves time and communication efficiency by over 30% compared to ARQ, while performing better than resource-intensive FEC-based schemes (HARQ). The key features of TVA, focusing error correction on only the most common error patterns, and injecting redundancy only when an error occurs, are likely to generalize well to future systems.

Future work includes integrating TVA with adaptive schemes. In adaptive schemes, ARQ is used in low-error situations, and FEC is used in high-error situations where more redundancy is needed. We believe that if TVA were substituted for ARQ in an adaptive system, it could greatly increase the amount of acceptable errors before a switch to FEC was required, improving performance when the BER is low. Another future direction is to investigate the potential applicability of TVA in the vast number of other systems where CRCs are used, such as mobile networks or RFID.

11. REFERENCES

[1] J. Cho and W. Sung. Efficient software-based encoding and decoding of BCH codes. *Computers, IEEE Transactions on*, 58(7):878 –889, July 2009.

[2] S.-M. Choi and B.-H. Moon. Implementation of energy efficient LDPC code for wireless sensor node. In T.-H. Kim, editor, *FGIT-FGCN (2)*, volume 266 of *Communications in Computer and Information Science*, pages 248–257. Springer, 2011.

[3] D. A. Eckhardt and P. Steenkiste. A trace-based evaluation of adaptive error correction for a wireless local area network. *Mob. Netw. Appl.*, 4:273–287, Dec. 1999.

[4] E. N. Gilbert. Capacity of a burst-noise channel. *Bell System Technical Journal*, 39:1253–1265, 1960.

[5] B. Han, F. Gringoli, and L. Cominardi. Bologna: block-based 802.11 transmission recovery. In *ACM S3*, pages 45–48, 2010.

[6] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller. Maranello: practical partial packet recovery for 802.11. In *NDSI*, pages 14–14, 2010.

[7] A. Iqbal, K. Shahzad, S. A. Khayam, and Y. Cho. CRAWDAD data set niit/bit_errors (v. 2008-07-08). http://crawdad.cs.dartmouth.edu/niit/bit_errors.

[8] K. Jamieson and H. Balakrishnan. PPR: Partial packet recovery for wireless networks. In *ACM SIGCOMM*, Kyoto, Japan, August 2007.

[9] J. Jeong and C. T. Ee. Forward error correction in sensor networks. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 2003.

[10] M. Kaur and V. Sharma. Study of forward error correction using Reed Solomon codes. *Int. Journal of Electronics Eng.*, 2(2):331–333, 2010.

[11] P. Koopman and T. Chakravarty. Cyclic redundancy code (CRC) polynomial selection for embedded networks. In *DSN*, pages 145 – 154, June 2004.

[12] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis. Surviving Wi-Fi interference in low power ZigBee networks. In *SenSys*, pages 309–322, New York, NY, USA, 2010. ACM.

[13] B. Lin. *Correcting Single-Bit errors with CRC8 in ATM cell headers.* Freescale Semiconductor, June 2005. Application Note AN2918.

[14] S. Lin, D. Costello, and M. Miller. Automatic-repeat-request error-control schemes. *Communications Magazine, IEEE*, 22(12):5 –17, December 1984.

[15] T. Mandel and J. Mache. Investigating CRC polynomials that correct burst errors. In *ICWN*, Las Vegas, Nevada, July 2009.

[16] T. Mandel and J. Mache. Selected CRC polynomials can correct errors and thus reduce retransmission. In *WITS (DCOSS)*, Marina Del Rey, CA, June 2009.

[17] B. McDaniel. An algorithm for error correcting cyclic redundance checks. *Dr.Dobb's Journal*, 2003.

[18] T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms.* John Wiley and Sons, New Jersey, 2005.

[19] D. Moss and P. Levis. *TEP 127: Packet Link Layer.* TinyOS Core Working Group.

[20] PCAUSA. *PCAUSA Test-TCP Utility (PCATTCP)*, August 2010.

[21] D. Schmidt, M. Berning, and N. Wehn. Error correction in single-hop wireless sensor networks: a case study. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1296–1301, 2009.

[22] Texas Instruments. *CC2420 Data sheet*, 2007.

[23] Texas Instruments. *CC2520 Data sheet*, 2007.

[24] M. Wines and M. Braathen. *Measuring the Power Consumption on the eZ430-RF2480.* Texas Instruments, 2008. Application Note AN057.