High-Efficiency Error Correction for Photon Counting

Andrew S. Fletcher

Pulse-position modulation (PPM) using a photon-counting receiver produces an extremely sensitive optical communications system, capable of transmitting multiple bits of information for each received photon. Such impressive sensitivity requires powerful error-correction codes that must be computationally efficient to enable high data throughput. Fountain codes combine performance and efficiency for a narrow class of channels, known as *erasure channels*. A potential application for fountain codes is the photon-counting PPM receiver, which is an approximate erasure channel and includes occasional channel errors. This non-ideal behavior requires a nontraditional use of fountain codes. With a carefully constructed architecture, fountain codes provide the desired efficient error correction to the photon-counting PPM receiver.

» **Today's data-driven society demands high** data rates and high-rate communication links. Addressing these needs in everything from gigabit Ethernet network connections and telephone or cable television signals to communication between spacecraft can require high bandwidth, high power, large transmit or receive apertures or both, and high receiver sensitivity.

Optical communication is attractive for these applications. High bandwidths are more readily available at optical wavelengths than at radio frequencies, and optical transmitters and receivers can be made compactly, as aperture size requirements scale with the signal wavelength. Highly sensitive photon-detecting receivers can be constructed to achieve high-rate data links capable of transmitting multiple bits per received photon. This exceptional receiver sensitivity requires an expanded bandwidth and powerful codes for correcting errors in the detected data. The bandwidth expansion is mitigated easily, but powerful error correction can be difficult to implement at gigabit-per-second data rates.

MIT Lincoln Laboratory is designing efficient forward error correction codes that combine several essential qualities, such as near-capacity rates, very low error floors, and efficient algorithms for coding and decoding. Such properties are desirable in nearly every communications system, but the efficient coding and decoding is particularly important for a high-data-rate, low-power system used in applications such as photon counting.

The photon-counting channel shares many characteristics with erasure channels. An erasure is a particular kind of error in which the receiver is unable to detect the transmitted symbol or somehow knows that the received symbol is wrong. In photon counting that uses pulse-position modulation (PPM), a type of transformation that encodes information in the precise timing of photon arrivals, the channel errors are overwhelmingly the result of dropped symbols that result in no signal detection. Only occasionally are there *dark counts*, which occur when the photon detector registers an arrival in the absence of any light and may result from thermal noise in detector electronics. Thus, the channel of interest is dominated by symbol erasures, with the rare case of a symbol error.

Recent results from the field of erasure channel coding may be leveraged to achieve code efficiency [1-4]. Symbol erasures accurately describe Internet packet transmissions; efficient use of network resources motivated erasure-correcting code design. A network-efficient solution is the fountain code, which is frequently used in Internet protocols and whose name describes the receiver's rateless nature. Fountain codes are one type of highly efficient error-correcting code for erasure errors and have several fantastic properties: they can be made arbitrarily close to the channel capacity, they can be decoded with a linear number of operations (proportional to the block length of the code), and the decoding can be performed with binary arithmetic rather than floating- or fixed-point operations.

The presence of occasional, but non-negligible, symbol errors complicates the use of fountain codes. The erasure-only decoder is not at all robust to the occasional symbol error, and robust decoders no longer enjoy binary-logic efficiency and likely require many more such operations (though still arguably a linear number) [5].

This work describes a binary-logic decoder for a raptor code (short for *rapid tornado*), a specific fountain code that is still robust to the occasional symbol error. In this way, the computational advantages of the fountain code decoder are retained and the decoder efficiency is sufficient to allow very high data rates.

Photon-Counting Pulse-Position Modulation

Highly-efficient, low-receive-power optical channels can transmit multiple bits of information for each received photon. To those unaccustomed to photon counting, this may appear impossible; it is readily explained via pulse-position modulation. In simple terms, the information is encoded in the precise timing of the photon arrival.

A PPM symbol consists of M time slots of duration t_{slot}, representing $\log_2 M$ bits of information. The transmitter places signal power in exactly one slot, leaving the

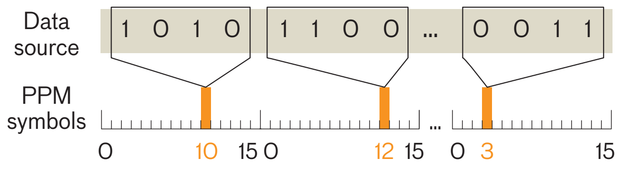

FIGURE 1. Each pulse-position modulation (PPM) symbol in the above example consists of 16 time slots, one of which contains a signal pulse. The four-bit message specifies which slot contains the pulse.

other slots for that PPM symbol empty. Figure 1 illustrates a 16-ary PPM in which each symbol represents four bits. The four-bit message is interpreted as a number between 0 and 15 and specifies the slot to contain the transmitted pulse. At the receiver, a photon counter determines which slot registered a photon arrival. If each symbol contains a single received photon (ignoring for a moment the statistics of real photon counting), then we have communicated $\log_2 M$ bits with each photon.

When we include some of the physical realities of photon counting, we see the similarity to an erasure channel. The receive signal is a coherent (laser) state with an intensity of N_S photons per symbol, localized to a single slot. With a 100% efficient photon counter, the actual number of received photons is a Poisson random variable with mean N_S. Thus, the probability of receiving zero photons in the signal slot is $\beta = e^{-N_s}$. For $N_S = 1$, this is approximately 0.37. Thus, despite averaging a photon per symbol, the receiver will have an empty symbol 37% of the time.

Channel capacity defines the maximum code rate at which error-free communication is possible; it is calculated as the mutual information between channel input and output (maximized over input distributions). Define $X \in \{1,\ldots,M\}$ and $Y \in \{1,\ldots,M,E\}$ as the random variables describing the transmitted and received symbols (with E indicating an erasure). Then $\Pr(X = x|Y = y) = 1$ and $\Pr(X = x|Y = E) = \Pr(X = x)$. The mutual information is given by $I(X;Y) = H(X) - H(X|Y) = (1-\beta)H(X)$. The capacity is achieved for equally likely input symbol X and is thus given by $C = (1-\beta)\log_2 M$—the capacity of the channel is the fraction of symbols that are not erased. The capacity calculated above assumes noiseless photon detection; however, there are several potential sources of noise, such as dark counts, background light (i.e., light not generated by the transmitter), and small amounts of light transmitted during supposedly empty slots. The noise is modeled by determining an average number of photons

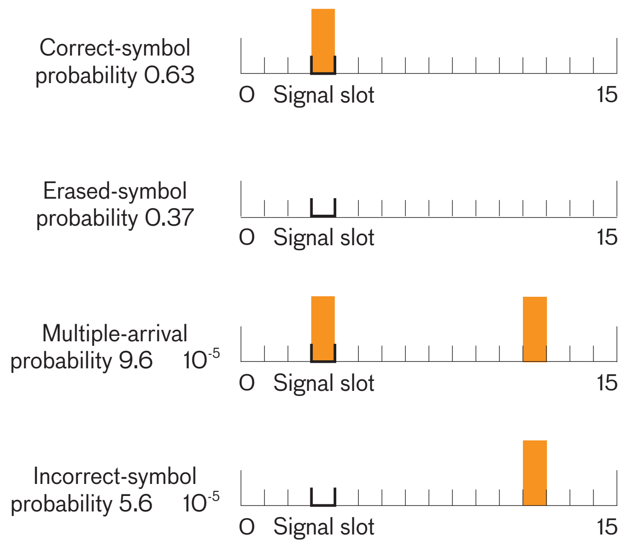
Correct-symbol probability 0.63
0 Signal slot 15
Erased-symbol probability 0.37
0 Signal slot 15
Multiple-arrival probability $9.6 \quad 10^{-5}$
0 Signal slot 15
Incorrect-symbol probability $5.6 \quad 10^{-5}$
0 Signal slot 15

FIGURE 2. **The photon-counting PPM receiver most often either detects a photon in the slot in which the transmitter placed the signal (the symbol is *correct*) or fails to detect a photon (the symbol is *erased*). Occasionally, the receiver detects a photon not generated by the transmitter. If the receiver also detects the correct signal, a *multiple arrival* occurs; if not, an *incorrect symbol* is registered.**

per slot N_B in which noise arrivals are independent of the transmitted signal. Cases are addressed for which N_B is quite small and corresponds to a receiver circumstance with negligible background light, a transmitter with a good extinction ratio, and a photon detector with a low dark-count rate. All of these assumptions represent realistic cases of interest.

Given such a model for dark counts, there are two noisy cases of interest. In the multiple-arrival case, the receiver observes a symbol with two photons observed in different time slots. While one photon was from the signal and the other from the noise, the receiver has no way of distinguishing them. A double arrival actually tells quite a bit about the transmitted signal, as the correct symbol corresponds to one of the two slots with high confidence. However, the decoder is simplified if the double-arrival case is merely flagged as an error and thus erased. The more problematic case is when a noise photon is observed during a symbol for which the signal photon does not arrive. This scenario results in a random symbol error that is not immediately discernible from a correct symbol.

For context, consider an example system based on technology under development at Lincoln Laboratory. A receiver with a 10 GHz slot rate, 16-ary PPM, and a half-rate code will operate at 1.25 gigabits per second (Gbps) throughput. If the received signal slot averages $N_s = 1$ photon, the receiver would obtain two bits per received photon. Four types of symbol detection are illustrated in Figure 2. The most common results are "correct" symbol detection, in which the photon is detected in the slot in which the transmitter placed the signal, and an erased symbol. For typical noise rates, a 1 kHz background photon arrival rate, 1 kHz dark-count rate, and 50 dB transmitter extinction ratio, the erasure probability (β) would be 0.37 and the probability of a correct detection would be 0.63; an erasure, 0.37; multiple arrivals, 9.6×10^{-5}; and incorrect symbols, 5.6×10^{-5}.

Error-Correcting Codes

An error-correcting code protects information in the channel by providing redundancy. A vector of information $\boldsymbol{x}$ is converted to a code word $\boldsymbol{y}$ by the encoder. The decoder takes the (channel-corrupted) code word and determines the best estimate for the original message $\boldsymbol{x}$. Codes are designed and defined to facilitate both encoding and decoding.

Fountain codes are linear codes, which are defined either in terms of a generating matrix $\mathbf{G}$ or a parity check matrix $\mathbf{H}$. The generating matrix defines the encoding operation, as the code word is defined as $\boldsymbol{y} = \mathbf{G} \times \boldsymbol{x}$ (using modulo-two multiplication). The decoder exploits the structure of the code word, particularly the parity check relation $\mathbf{H} \times \boldsymbol{y} = 0$. Commonly, the code is defined by either $\mathbf{H}$ or $\mathbf{G}$, with the other a derived quantity.

Fountain codes are typically low density, which means that a small fraction of entries in either $\mathbf{H}$ or $\mathbf{G}$ are nonzero. In general, either $\mathbf{H}$ or $\mathbf{G}$ can be low density, but not both.

Figure 3(a) uses a bipartite graph to illustrate a generating matrix for one code; Figure 3(b), a parity check matrix for a different code. In both graphs, the nodes on the left represent bits y_i of a code word $\boldsymbol{y}$. In Figure 3(a), nodes on the right represent parity checks, in which the modulo-two sum of the bits connected to each check is zero. In Figure 3(b), the nodes on the right represent the information to be encoded and left nodes are the modulo-two sum of the associated right nodes.

Erasure decoding is a straightforward process for both low-density parity check (LDPC) and low-density generating matrix (LDGM) codes. In both cases, the code

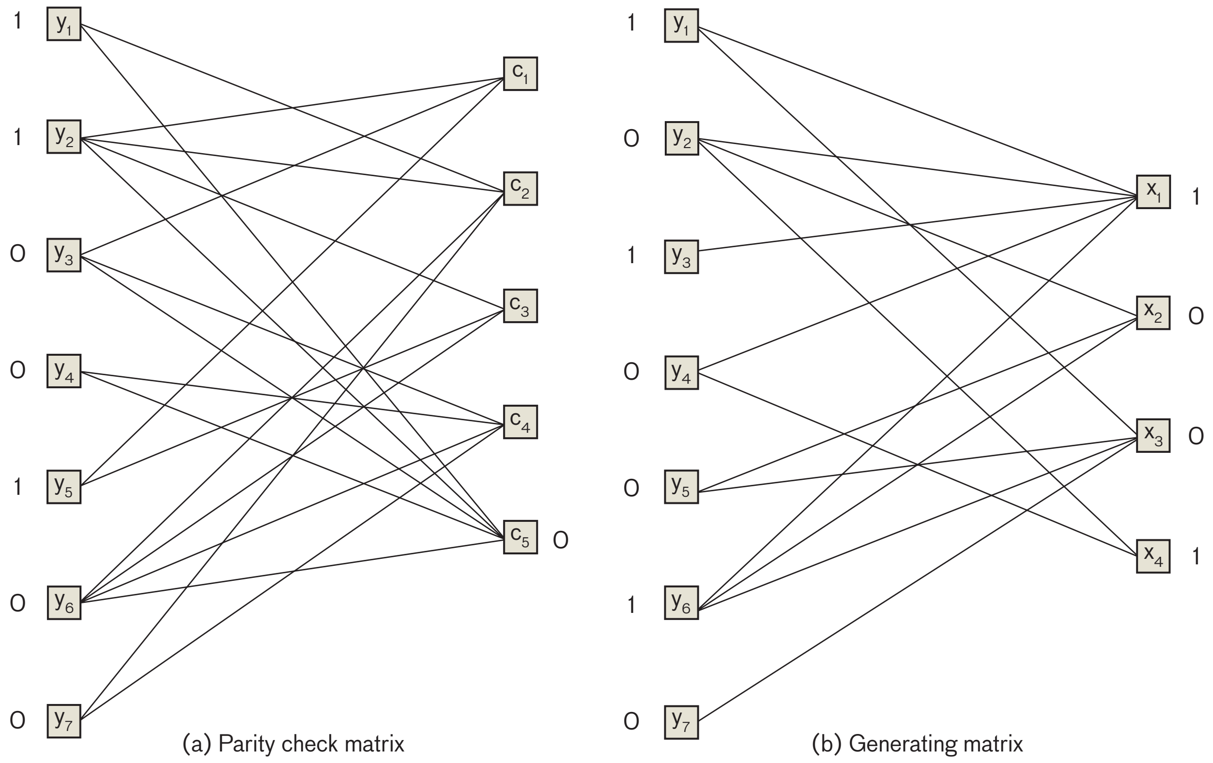

FIGURE 3. Bipartite graphs representing (a) a parity check matrix **H** and (b) a generating matrix **G** are shown for two different codes. For both graphs, the nodes on the left represent the bits y_i in a code word. For (a), nodes on the right (labeled c_j) represent parity checks; the modulo-two sum of the bits connected to each check is 0. For (b), nodes on the right (labeled x_j) represent the information to be encoded. Left nodes are the modulo-two sum of the associated right nodes.

enforces a binary structure on the received bits. Consider the i^{th} row of **H**, with nonzero entries $\{i_1, i_2, \dots i_k\}$. The parity equation requires that

$$0 = y_{i_1} \oplus y_{i_2} \oplus y_{i_3} \oplus \dots \oplus y_{i_k},$$

in which $\oplus$ indicates modulo-two addition. If exactly one of the $\{y_{i_1}, y_{i_2}, \dots, y_{i_k}\}$ is erased, the parity relation unambiguously would supply the unknown value. Similarly for the LDGM, a row of **G** leads to a requirement that

$$y_{i_1} = x_{i_1} \oplus x_{i_2} \oplus x_{i_3} \oplus \dots \oplus x_{i_k}.$$

If y_i and $k - 1$ of the x_i are known, then the unknown k^{th} bit can be determined unambiguously. The parity or generating equations suggest a simple iterative decoding algorithm using either **H** or **G**, respectively. At each iteration, locate a row of **H** or **G** that contains only one unknown variable. The code structure provides the unknown variable's value, which is then used to update the values of connected nodes. The connections are severed, and the next iteration identifies another row with only one unknown variable. The decoding continues until all variables are known (a successful decoding) or there are no rows with only one unknown (a decoding failure).

The iterative erasure decoding is sometimes described as a *chain reaction decoder*, and it is straightforward to understand graphically. Figure 4 illustrates two iterative decodings for the generating and parity-check matrices shown in Figure 3.* The decoding in Figure 4 represents the parity-check graph of Figure 3(a) in which the y_2, y_3, y_4, and y_5 bits have been erased. The erased bits are assigned to the right nodes ρ and the parity check nodes, c_i, are assigned to the left nodes, λ. The initial values in λ have been chosen to yield the appropriate parity-check values for combinations of the unerased bits y_1, y_6, and y_7. To determine the values of the unknown bits, the decoder identifies code dependencies with only one unknown, which correspond to left nodes with only one edge (a degree-one node). In the

* The two graphs in Figure 3 lead to the same decoding procedure by design and for ease of explanation. Recall that Figures 3(a) and 3(b) describe different codes and that different bits have been erased in each message.

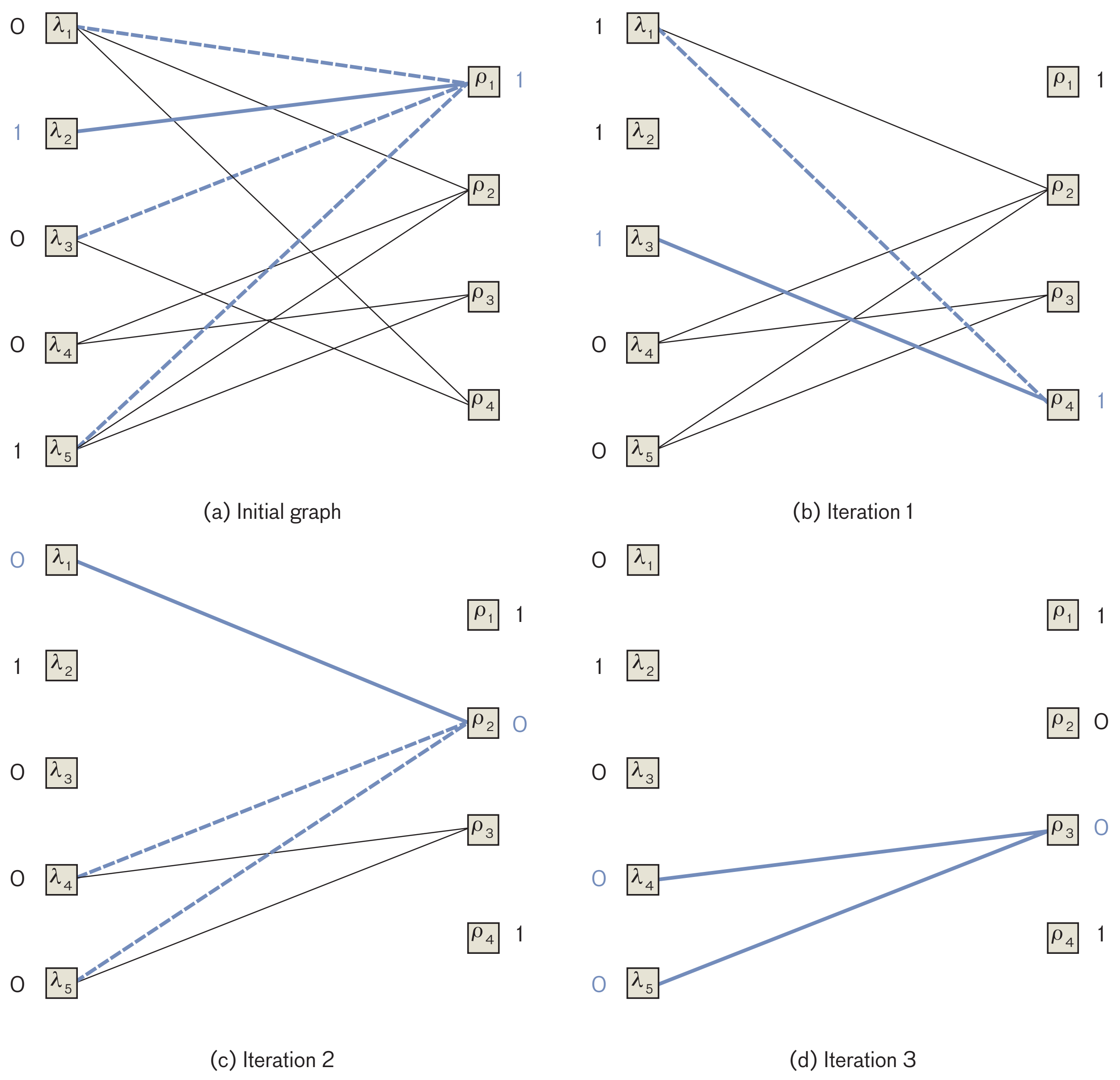

FIGURE 4. A graphical example is presented of binary-message erasure decoding. The initial graph (a) is shown with the right-hand nodes (ρ_i) indicating the bits to be filled and the left-hand side (λ_i) denoting the values supplied by non-erased symbols. Values of λ drawn in blue are used to determine values of ρ that are also drawn in blue. Edges shown in blue are then used to propagate the newly determined values of ρ. In iteration 1 (b), node ρ_1 is filled in as having value 1, which then is added (modulo two) to the values in λ_1, λ_3, and λ_5, and the linking edges are removed. In a similar fashion, ρ_2 is determined in iteration 2 (c), ρ_3 in iteration 3 (d).

initial graph, λ_2 is a degree-one node; thus, ρ_1 can be assigned the value of λ_2 (1) and the edge linking ρ_1 and λ_2 can be removed. Next, ρ_1 is added (modulo two) to the value of any other left node to which it is connected and those edges are removed. Thus, λ_1, λ_3, and λ_5 are each updated with the modulo-two addition of ρ_1, resulting in Figure 4(b). For the next iteration, λ_3 is identified as a degree-one node. The value of ρ_4 is determined (1) and added to connected nodes, and the graph is updated to produce Figure 4(c). For the next iteration, λ_1 is identified as a degree-one node, ρ_2 is determined and added to connected nodes, leading to Figure 4(d). The value of ρ_3 is trivial here, supplied by either λ_4 or λ_5. The values of ρ_1 through ρ_4 match the source bits in Figure 3(a), confirming that this decoding has produced the correct values for the deleted bits.

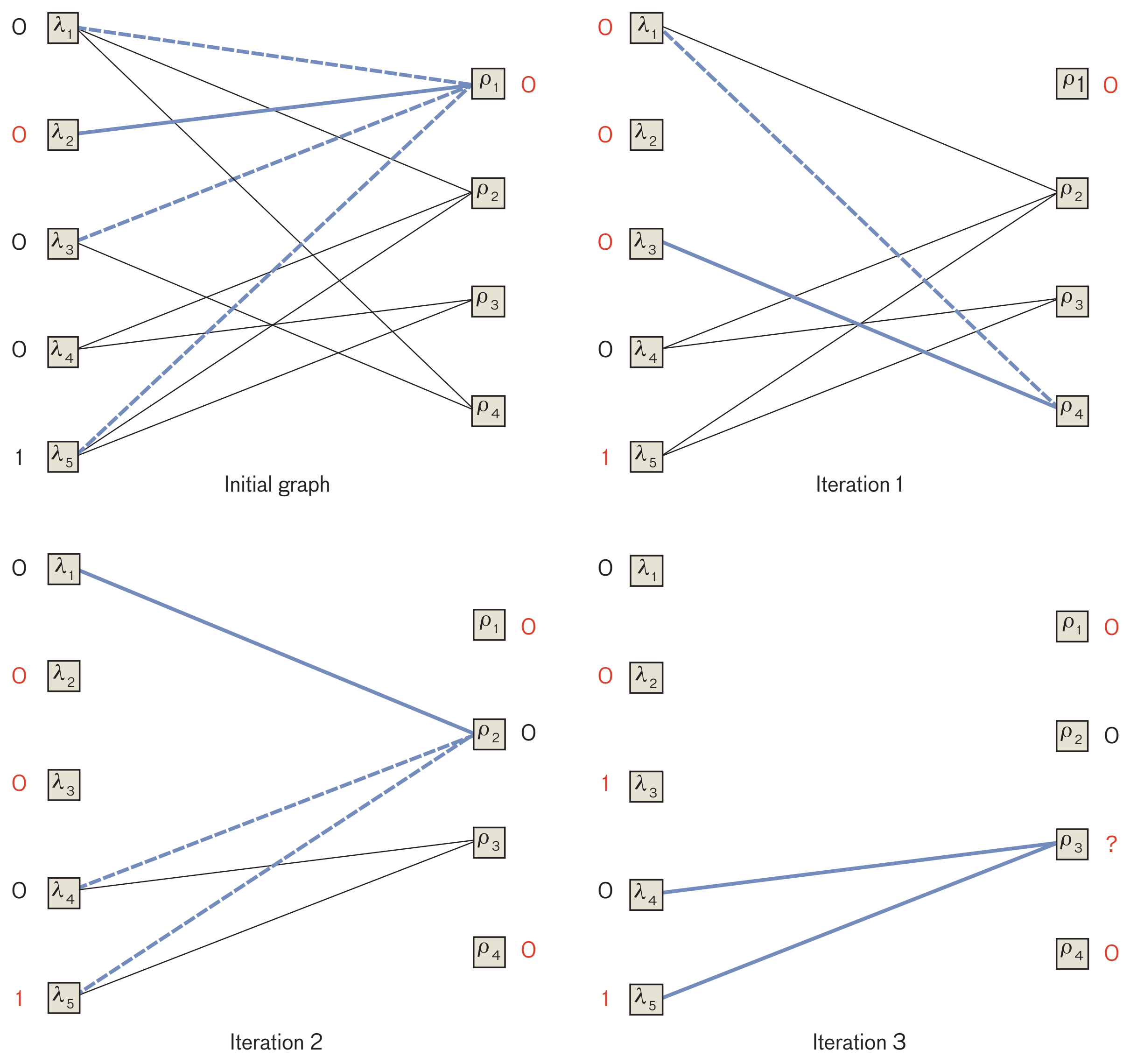

FIGURE 5. A graphical example of binary-message erasure decoding in the presence of a bit error. The same decoding process is used as in Figure 4, but the bit error leads to incorrect results. Node λ_2 is assigned the wrong bit value. As the decoding progresses, red values indicate a departure from the correct decoding. The decoded sequence is wrong in bits ρ_1 through ρ_4. Note also the contradictory choice for the value of ρ_3.

The graphical representation of decoding is equally effective with the generating matrix of Figure 3(b). This matrix also has a seven-bit code word, and if y_1 and y_7 are erased by the channel, the four message bits must be determined from the five unerased bits. The initial graph in Figure 4(a) is simply the graph in Figure 3(b) with the y_1, y_7, and the attached edges removed. The left and right nodes are also relabeled as λ and ρ, respectively. The same iterative procedure described above may be used to determine the erased bits: identify a degree-one node, determine the value of ρ_i from its one-degree connection to λ_j, add the determined value of ρ_i to all other connected left nodes and remove the edges, find the next degree-one node, and repeat.

Graph design requires careful consideration of degree-one nodes. As illustrated above, decoding requires at least one degree-one node to continue, but if there are too many such nodes at any iteration, the code is inefficient. Finding the appropriate balance has been the subject of extensive research [3, 4, 6–9].

The most successful fountain codes fall under the classification of raptor codes [3, 4]. These are, in fact, two-stage codes. The inner code is an LDGM code, while the outer code is a very-high-rate LDPC code. The raptor code approaches capacity and has excellent error floors while maintaining a linear number of graph edges. Because the number of decoding computations is proportional to the number of graph edges, the raptor code is a particularly efficient choice for the erasure channel.

Effect of Errors on Chain-Reaction Decoding

Although the chain-reaction decoder is computationally simple, it is only robust to channel erasures. A single-bit error in the receive data can lead to a catastrophic decoding failure. For example, in the LDGM decoding example in Figure 4, y_1 and y_7 were erased. If instead, a bit error is included on y_3, corresponding to a bit error on λ_2, the decoding procedure will result in errors. As shown in Figure 5, the decoded message becomes 0000 instead of the correct 1001. In general, a bit error on a node can propagate through the decoder to cause errors on many decoded bits.

The potentially catastrophic impact of a few errors is clearly a black mark against chain-reaction decoding for the noisy photon-counting PPM channel and could limit its promise for high-data-rate, highly efficient optical signaling. Typically, the channel of interest is dominated by erasure errors with only the occasional dark count causing a symbol error. In these situations, the use of the chain-reaction decoder should not be abandoned; although the chain reaction decoder does not correct transmitted errors, it does detect errors with high probability. At the last step in the example of Figure 5, the bit error led to a contradiction; such inconsistencies only arise because of errors. The error-detection capability is an important element of the proposed photon-counting PPM code architecture.

Code Architecture for Photon-Counting PPM

An alternative approach combines raptor codes and chain-reaction decoding and constrains the size of code words to improve robustness to errors. For an [n; k] raptor code that encodes a k-bit message into an n-bit code word, after passing through a photon-counting PPM channel, bits are erased with probability β and are in error with probability γ, in which $\gamma << \beta$. Because the capacity (the upper bound for the code rate) of this channel is approximately $1 - \beta$, a code rate of $k/n < 1 - \beta$ is required.

Two situations cause the chain-reaction decoder to fail: if (at least) one of the n bits is an error or if there are too many erasures. For small γ, the probability that the code word sees an error is approximately $n\gamma$. An obvious way to keep this probability small is to limit the block length n.

What is the probability that there are too many erasures? For a perfect code, we are able to determine the message if any k (or more) of the bits arrives unerased. A realistic code, such as a raptor code, is decoded successfully when more than $k + n\varepsilon$ bits arrive unerased, in which ε represents a code overhead and implies a distance from capacity.* Raptor codes are capacity-achieving for code families in which ε approaches zero as n approaches one. As n gets small, the required overhead ε grows.

Finite-length codes are always subject to decoding failures. For an erasure channel, each bit is independently erased with probability β. The number of erasures N_E that occur in an n-bit code word is a binomial random variable with mean $n\beta$ and variance $n\beta(1 - \beta)$. A decoding failure occurs if $N_E/n > 1 - k/n - \varepsilon$. For large n, the law of large numbers suggests that decoding failures are quite rare. For smaller n, however, decoding failures become more likely.

The two conditions described that can result in a failure of the chain-reaction decoder present competing requirements for limiting the probability of such a failure. On the one hand, as the block length grows, the code word is more likely to be corrupted by an error. On the other hand, raptor code performance improves as n increases. Fortunately, a useful operating point is contained within this trade space. Figure 6 illustrates this trade-off for a raptor code with overhead $\varepsilon = 0.1$, $\beta = 0.37$, and a symbol-error probability of $\gamma = 5.6 \times 10^{-5}$. For block sizes between 600 and 1400, the probability of a decoding failure is below 0.08.

The output of the chain-reaction decoder is a k-bit block erasure channel. If the decoder is successful, the k-bit message is successfully determined; if the decoder fails because of an error or too many erasures, the failure is detected and no message is returned. As long as decoder failures are always detected, the raptor code and chain-reaction decoder combination results in a purely erasure

* Strictly speaking, when $k + n\varepsilon$ bits arrive unerased, decoding is successful *with high probability*.

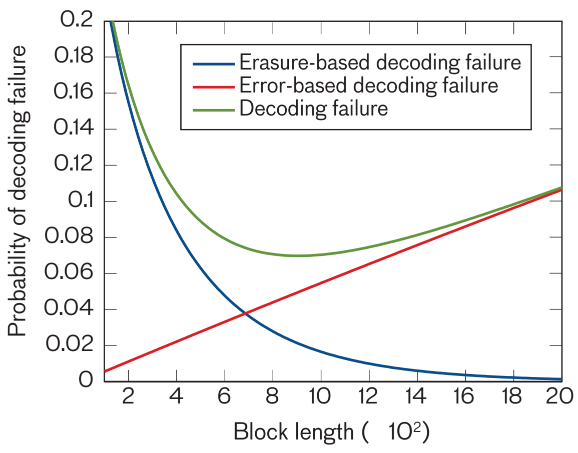

FIGURE 6. A trade-off exists between raptor-code block length and the probability of a decoding failure. For a half-rate raptor code with an overhead $\varepsilon = 0.1$, erasure probability $\beta = 0.37$ and a symbol error probability of $\rho = 5.63 \times 10^{-5}$, the chain-reaction decoder fails in the presence of an error or too many erasures.

channel, in which the erasure probability is the probability of a decoding failure.

The improved code architecture combines an inner raptor code having a rate $1 - \beta - \varepsilon$ and a block length n chosen to minimize the probability of decoder failure, and an outer code that can be any erasure correcting code in which k bits are encoded for every node. The outer code is a relatively high-rate code, as it need only correct the small fraction of blocks on which the inner code fails. A block diagram for this system is shown in Figure 7. This code structure achieves a highly sensitive receiver with a very high data throughput.

Serial Versus Parallel Chain-Reaction Decoders

The algorithm description introduced for the chain-reaction decoder above implies a serial decoder implementation. Functionally, the serial decoder maintains a list of all degree-one left nodes. At the beginning of each iteration, a node from this list is used to fill in the attached message node; next, the left node is removed from the list. All of the remaining left nodes attached to the now-filled message bit are appropriately updated. If the update results in a new degree-one node, it is added to the list. If the update results in a degree-one node becoming degree zero (because it was also attached to the now-filled message node), it is removed from the list. The serial decoder proceeds until the degree-one list is empty.

The chain reaction succeeds by performing an iteration for every unknown message bit. Thus, the serial implementation is often appropriate for cases in which the frequency of erasures is low. When the erasure frequency is high, a parallel implementation of the chain-reaction decoder may be desirable. At any point in the chain reaction, there may be several degree-one left nodes. Recognizing this, during each iteration, the parallel decoder may process values from all the degree-one nodes. Thus, each message node determines if it is degree one; if so, it sends its value to the attached message nodes. These messages between the left and right nodes constitute a single iteration. In the parallel implementation, the entire code word could be determined in a relatively small number of iterations. For the codes of interest, it often requires 50 to 100 iterations regardless of the number of bits in the code word. In contrast, the serial decoder requires an iteration for every erased bit, which can number in the thousands.

The parallel decoder differs markedly from the serial decoder. If the algorithm were laid out as a circuit (e.g., in an application-specific integrated circuit or a field-programmable gate array [FPGA]), every node would occupy a particular location and would have a physical connection to other attached nodes. Despite the sparse nature of

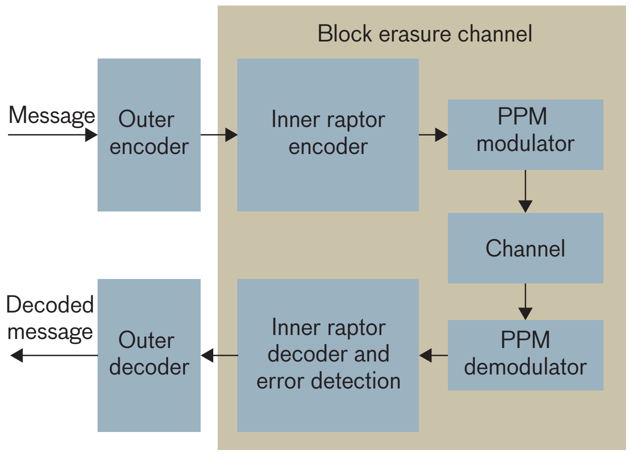

FIGURE 7. A code architecture block diagram illustrates a message traversing an outer encoder, a block erasure channel, and, finally, an outer decoder. The inner raptor code, which has a short block length, surrounds the noisy photon-counting channel. When a symbol error occurs or when the inner decoder fails, the inner code word is dropped. The overall effect is a block erasure channel, in which the erasure probability is kept low. The outer code is employed to correct the block erasures.

the graph, these connections can become dense. Also, the graph should be random or pseudorandom for good performance—highly structured graphs often perform poorly with the chain-reaction decoder. Furthermore, every left node must compute whether it is weight one. For high-degree nodes, this computation can limit the clock speed of the algorithm. These challenges for the parallel decoder make its implementation difficult unless the block size is small. One should also note that if a graph node represents k bits, then the physical connection between the nodes must be a bus with width k.

The strengths and limitations of serial and parallel decoding align themselves well with the photon-counting PPM architecture. The inner raptor code sees a high rate of erasures, but is restricted to small block sizes. As such, the inner code is ideally suited for a parallel decoder. The parallel decoder greatly enhances the overall system throughput, as multiple bits are decoded for every computational clock cycle. The outer code has a high rate and sees infrequent erasures. Furthermore, each node represents the k message bits from the inner code. The outer code is ideally suited for a serial decoder.

Interestingly, the hardware requirements are highly complementary. The parallel decoder requires significant real estate on an FPGA, but requires essentially no memory. The serial decoder requires almost no real estate, but all of the graph connections and the list of degree-one nodes is maintained in memory.

Short-Block Raptor Design

The proposed code architecture requires a raptor code with both a low code overhead (i.e., a small ε) and a short block length (e.g., $n \approx 1000$). Since their introduction, most research in designing fountain codes (including raptor codes) has focused on large block sizes [1–4]. Although considerably less attention has been devoted to designing high-performance raptor codes with short block lengths, some short-block ideas have been leveraged to design the implemented raptor codes [10–12].

Raptor codes, as with most fountain codes, are random. Instead of designing a specific graph, a raptor code design specifies the statistical properties of a graph. The random structure allows a statistical analysis of the code performance; one can derive a mathematical expression for the probability of successful decoding.

Statistical code analysis is most easily performed and understood on random LDGM codes called Luby Transform (LT) codes after their inventor Michael Luby [4]. Raptor codes are LT codes with a small (but significant) modification, so raptor code design begins with LT code design.

A good code design is one with a high probability of a successful chain-reaction decoding. Consider a k-bit message encoded in an LT code word, of which m bits arrive unerased at the receiver. The resulting graph has the structure of the LDGM graph in Figure 4 and has a simple statistical description. There are m left nodes and k right nodes. Each left node has D edges attached, in which D is a random variable with probability mass function Ω_d. (That is to say, the probability that a left node has d attached edges is Ω_d.) The D edges are randomly attached to the right nodes in a uniform way, independent of all of the other left nodes. With this description, an LT code is completely defined by selecting a degree distribution Ω_d.

Selecting Ω_d is done by calculating the probability of successfully decoding such a random graph. For very large code words (e.g., $k > 50{,}000$), this has been typically done by computing the probability of decoding in the limit as k goes to infinity (and $m = k(1+\varepsilon)$). For short-block code design, in which k is on the order of 1000, the decoding probability can be calculated using a dynamic program.

A dynamic program is a method to solve a complex problem by dividing it into simpler steps. At each step, a set of values is computed; these values are used and updated in subsequent steps. For the LT code, the program is divided into the k iterations necessary to successfully complete a serial chain-reaction decoder. The precise mathematical derivation is beyond this article's scope, but the basic concept is relatively straightforward. The chain-reaction decoder requires at least one left node of degree one to continue the decoding process. A serial decoder selects one such degree-one node and determines the value of the attached right node. It takes exactly k steps to successfully complete the decoder, and k steps will occur if (and only if), after each iteration, at least one degree-one node remains.

As shown in Figure 8, before the first decoding iteration, the left nodes can be divided into two sets: the degree-one nodes and the nodes having degree greater than or equal to two ("two-plus nodes"). At each iteration, a single node from the degree-one set becomes degree zero after it is used to determine the value for a right node.

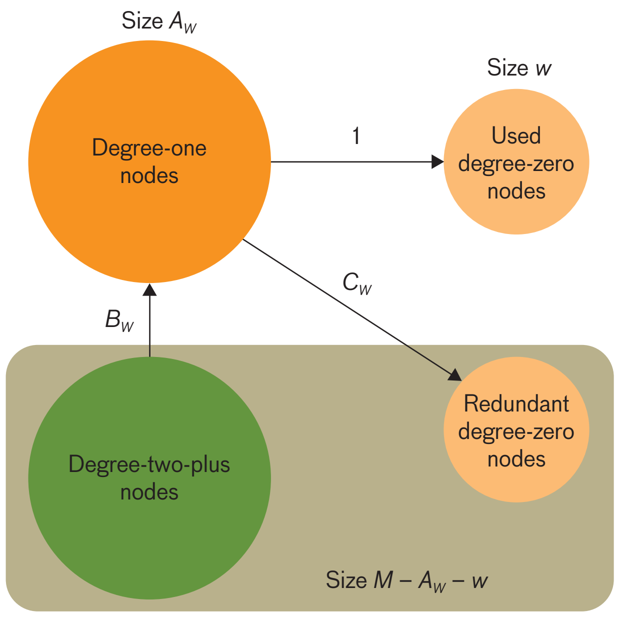

FIGURE 8. After w right nodes have been decoded, the left nodes are divided into sets according to their degree: zero, one, or greater than or equal to two ("two-plus"). Degree-zero nodes are further divided into nodes used in the decoding and nodes that are redundant. Assuming at least one degree-one node remains, it is used by the decoder, B_w degree-two-plus nodes become degree-one, and C_w degree-one nodes become redundant degree-zero nodes.

This degree-zero set has a size equal to w. During the same iteration, it is possible for other degree-one nodes to become degree zero (e.g., step 3 of the example scenario in Figure 4, in which two degree-one nodes are attached to the same right node). It is mathematically useful to distinguish between these two sets of degree-zero nodes: the first set is used in the decoding, while the second set is redundant. The size of this redundant degree-zero set of nodes is denoted by C_w.

The most important set is the set of degree-one nodes; the number of degree-one nodes after w iterations is a random variable A_w. There are exactly w decoding degree-zero nodes, one for each iteration of the decoder. The remaining two sets contain the redundant degree-zero nodes and the two-plus nodes. Together, these sets contain $M - A_w - w$ nodes. At each iteration (assuming there is at least one degree-one node), exactly one node transitions from degree one to degree zero, while B_w nodes transition from the two-plus set to the degree-one set, and C_w nodes transition from the degree-one set to the redundant degree-zero set. Thus, the number of degree-one nodes after the $w + 1$ iteration is computed as $A_{w+1} = B_w - C_w - 1$.

These observations provide a straightforward recipe for computing the probability of successful decoding. The decoding succeeds if $A_w > 0$ for all $w = 0 \ldots k - 1$. The initial distribution of A_0 is a simple function of Ω_0; the dynamic program updates the distribution of A_w and updates it after each iteration. The algorithm updates the distribution by evaluating the random variables B_w and C_w, which illuminate how A_w evolves. The key feature of the algorithm is the independence of each random variable. Conditioned on A_w, B_w and C_w are independent from one another and from all previous decoding steps. The dynamic program must also maintain and update the distribution of node degrees in the evolving sets of two-plus nodes and redundant zero-degree nodes (both of which are highlighted in Figure 8).

The dynamic program provides a probabilistic score for a distribution Ω_d. It computes the probability that the decoder fails at each iteration; the total probability of failure is the sum of these probabilities. This dynamic program is an ideal tool for designing LT codes of relatively short length, but lacks the final modification necessary to design raptor codes.

Raptor codes were invented by Shokrollahi to overcome a known deficiency in LT codes [10–11]. LT codes require k iterations to successfully decode. It is easy to design an LT code with low average degree (and hence an efficient decoder) that will successfully complete most of the k iterations (approximately greater than 95%). Decoding the final few iterations, however, requires a significant expansion of the number of graph edges. The raptor code starts with an efficient LT code and adds a *pre-code* to handle the last few iterations. With this computation, the total number of edges in the graph remains small (proportional to the message size k).

To design a good LT code for use in a raptor code, the dynamic program requires a small adjustment to account for the pre-code. If the LT code fails after $w < k$ iterations, the pre-code may successfully complete the decoding. Fortunately, the dynamic program can be modified to include such calculations in a straightforward way.

Ongoing Work

The fountain code architecture for photon-counting PPM is, at its roots, a practical compromise based on current hardware realities. Raptor codes have a straightforward decoder implementation that enables high-throughput communications without overwhelming computational resources. Because the code architecture is designed for practical implementation, it is well suited to a technology demonstration, which is currently ongoing at Lincoln Laboratory.

As mentioned above, the code structure has places for both a parallel (for the inner code) and serial (for the outer code) chain-reaction decoder. In the ongoing demonstrations, both decoders have been programmed in hardware description language for implementation on an FPGA device. An FPGA combines some of the parallelism and computational throughput of a custom-designed chip with the flexibility of a programmable device. In the current experiments, the inner and outer decoders are each implemented on separate Virtex-5 SX240 FPGAs; in future implementations, these may be combined into a single FPGA. Both the inner and outer encoders are implemented on a smaller Virtex-5 (LX110) FPGA. Preliminary results show that the decoder can operate in this mode at upwards of 1 Gbps data rates with a rate ½ code. Such a code should achieve nearly two bits per received photon.

The eventual demonstration will incorporate the encoder and decoder into an experimental photon-counting receiver. Researchers at Lincoln Laboratory have built a photon-counting receiver test bed that uses innovative superconducting nanowire single-photon detectors. The test bed has been used to demonstrate 2.5 Gbps operation, but the demonstration required offline processing for error correction. Real-time decoding demonstrations have, until now, been limited to 100 Mbps throughput. The photon-counting receiver test bed is an ideal platform for demonstrating throughput capabilities of the chain-reaction decoder.

Acknowledgments

Many thanks to Don Boroson and Bryan Robinson for their help in the original coding concept. Thanks also to Matthew Willis and Rohit Kochhar for work in hardware implementation. ■

REFERENCES

1. M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inform. Thy.*, vol. 47 , no. 2, 2001, pp. 569–584.
2. M. Luby, "LT codes," *Proc. 43rd Annual IEEE Symp. Foundations of Comp. Sci.*, 2002, pp. 271–280.
3. M.A. Shokrollahi, "Raptor codes," *Proc. Intl. Symp. Inform. Thy.*, 2004, p. 36.
4. M.A. Shokrollahi, "Raptor codes," *IEEE/ACM Trans. Netw., Special Issue on Networking and Information Theory*, vol. 14, 2006, pp. 2551–2567.
5. O. Etesami and A. Shokrollahi, "Raptor codes on binary memoryless symmetric channels," *IEEE Trans. Inform. Thy.*, vol. 52, no. 5, 2006, pp. 2033–2051.
6. M.A. Shokrollahi, "Applied algebra, algebraic algorithms and error correcting codes," *Lecture Notes in Computer Science, New Sequences of Linear Time Erasure Codes Approaching the Channel Capacity*, vol. 1719/1999, Berlin/Heidelberg: Springer, 1999, pp. 65–76.
7. M.A. Shokrollahi and R. Storn, "Design of efficient erasure codes with differential evolution," *Proc. IEEE Intl. Symp. Inform. Thy.*, 2000, p. 5.
8. T.J. Richardson, M.A. Shokrollahi, and R.L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Thy.*, vol. 47, no. 2, 2001, pp. 619–637.
9. C. Di, D. Proietti, I.E. Telatar, T.J. Richardson, and R.L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Thy.*, vol. 48, no. 6, 2002, pp. 1570–1579.
10. R. Karp, M. Luby, and A. Shokrollahi, "Finite length analysis of LT codes," *Proc. Intl. Symp. Inform. Thy.*, 2004, p. 37.
11. E. Maneva and A. Shokrollahi, "New model for rigorous analysis of LT-codes," *IEEE Intl. Symp. Inform. Thy.*, 2006, pp. 2677–2679.
12. A. Venkiah, *Analysis and Design of Raptor Codes for Multicase Wireless Channels*, Ph.D. thesis, University of Cergy-Pontoise, 2008.

ABOUT THE AUTHOR

Andrew S. Fletcher is a technical staff member in the Optical Communications Technology Group. His work focuses on error correction and information theory for optical communication systems. He earned both a bachelor's and master's degree from Brigham Young University in 2001, as well as a doctorate degree from Massachusetts Institute of Technology in 2007, all in electrical engineering.