

# Online Coding for Reliable Data Transfer in Lossy Wireless Sensor Networks

Anthony D. Wood and John A. Stankovic

Department of Computer Science  
University of Virginia  
{wood,stankovic}@cs.virginia.edu

**Abstract.** Bulk transport underlies data exfiltration and code update facilities in WSNs, but existing approaches are not designed for highly lossy and variable-quality links. We observe that Maymounkov’s rateless online codes are asymptotically more efficient, but can perform poorly in the WSN operating region. We analyze and optimize coding parameters and present the design and evaluation of RTOC, a protocol for bulk transport that recovered over 95% of application data despite up to 84% packet loss in a MicaZ network.

## 1 Introduction

Often wireless sensor networks (WSNs) must reliably transfer large amounts of data, which is challenging given the typical resource constraints of WSN devices. They may be deployed in adverse circumstances where *poor and highly variable link quality* is caused by dynamic environmental factors such as heat and humidity, by low-cost hardware and its concomitant failure or unreliability, or by obstacles and RF interference (accidental or malicious). Whether for extracting sensor data or loading new code in over-the-air reprogramming, bulk data must be transmitted *efficiently* to reduce wasted computation and communication. These twin problems of loss-tolerance and efficiency are not sufficiently addressed by the state of the art.

Existing protocols use various methods to conceal or overcome loss of data blocks. The approaches taken by Deluge [1], RCRT [2], and Flush [3] are based on Automatic Repeat Request (ARQ), in which ACKs or NACKs explicitly request retransmission of lost data. However, in severe conditions ARQ protocols require many retransmissions and have high latency, as determined by Kumar [4] for TCP in lossy networks.

Another pragmatic approach to achieving reliability in this setting is to bound the expected error rate  $\delta$  and use forward error correction (FEC) for transmitting blocks of the data. For predictable channel conditions, a code may be chosen that is a trade-off between overhead and performance, and it has been proven that codes exist with rate equal to the channel capacity  $1 - \delta$ . However,

---

This work was funded in part by ARO grant W911NF-06-1-0204, NSF grants CNS-0614773 and CNS-0614870.

under intermittent interference or other lossy conditions, the channel may be arbitrarily bad, and for any error rate greater than  $\delta$ , a fixed-rate code fails completely and the block or message is lost. Pessimistically chosen parameters suffer from high overhead which must always be paid.

These limitations motivate the use of rateless erasure codes, also called fountain codes. They have recently attracted attention for use in WSNs [5, 6] due primarily to these properties: first, a limitless number of encoded symbols can be generated from an input of  $k$  symbols, and second, the original  $k$  symbols can be recovered from any  $k' = (1 + \epsilon)k$  received encodings (asymptotically, for fixed  $\epsilon$ ). Theoretically, no feedback channel is needed, such as for the ACKs and NACKs of an ARQ protocol. The sender can transmit an endless stream of encoded symbols and even for an arbitrarily poor channel (as long as  $\delta < 1$ ), the receiver eventually receives  $k'$  symbols and can decode the message. Such an encoding scheme is optimal if  $k' = k$ .

Rateless Deluge [5] uses random linear codes which require multiplication modulo an irreducible polynomial to decode. Extra memory is needed for storing inversion tables to achieve practical execution speeds, and both encoding and decoding are complex for typical WSN platforms. Luby's LT codes [7], which are the basis of SYNAPSE [6], more efficiently encode packets using exclusive-OR operations at the sender, but also require Gaussian elimination for decoding, which cannot proceed until  $k$  blocks are received.

We propose the use of online codes [8], which improve on LT codes to achieve  $O(1)$  time encoding (per block) and  $O(n)$  decoding, and which permit iterative decoding as packets are received. However, coding parameters recommended for Internet networks perform poorly in messaging overhead and memory consumption in the typical WSN operating region of relatively few data blocks. This prevents their direct replacement in existing protocols, and motivates this study.

This work uses online codes to provide reliable data transfer despite highly lossy communication channels. To do so, it must address challenges in the selection of appropriate parameters for the coding scheme and requires a protocol design that minimizes round-trip interactions. Our contributions include:

- We design Reliable Transfer with Online-Coding (RTOC), a novel transport protocol for WSNs that is the first to employ online codes for higher decoding efficiency than SYNAPSE. It stays synchronized despite high loss rates, and uses feedback control to adaptively terminate data transmission without ARQ as in Deluge or manual FEC selection used by Rateless Deluge.
- Through analysis of the online coding degree distribution and algorithm, we optimize parameters to trade asymptotic optimality for predictability within the WSN operating region. We achieve a 12% better effective coding rate with 72% lower variance, which reduces the 98<sup>th</sup> percentile decoding memory requirements by 69%.
- We evaluate the performance of RTOC on an implementation in TinyOS for the MicaZ platform, and show that block delivery ratios exceed 95% despite up to 84% packet loss. Overhead follows from the page fragmentation and effective coding rate, and is low when channel loss is low.

In the next section we describe related work, and then present the design of our loss-tolerant transport encoding scheme in Section 3. Key coding parameters are analyzed in Section 4 for their impact on efficiency in WSNs. Evaluation of an implementation for MicaZ motes is given in Section 5. Finally, we conclude in Section 6.

## 2 Related Work

Methods exist for selecting high quality links to avoid poor communication [9] and for detecting inconsistencies to trigger code updates [10] in WSNs and are orthogonal to this work.

Over-the-air reprogramming has been addressed by other schemes that pre-date Deluge [1], but the latter has become a popular choice despite its shortcomings. Recent work has attempted to improve its efficiency and performance using rateless codes as described above [5,6]. RTOC builds upon this work and adopts some features common to reprogramming protocols, but is modularized to allow its use for other purposes such as bulk data transport, and nothing precludes its use as an underlying mechanism for code updates.

Flush [3] is an end-to-end transport protocol for WSNs that uses acknowledgments and rate control to achieve high goodput. Like RCRT [2], another rate-controlling transport protocol, it relies on round-trip messaging to drive the control algorithm. While this gives good performance at each hop when the channel loss is low, it performs poorly when many control messages are lost [4]. RTOC is designed to tolerate such losses in its feedback mechanism.

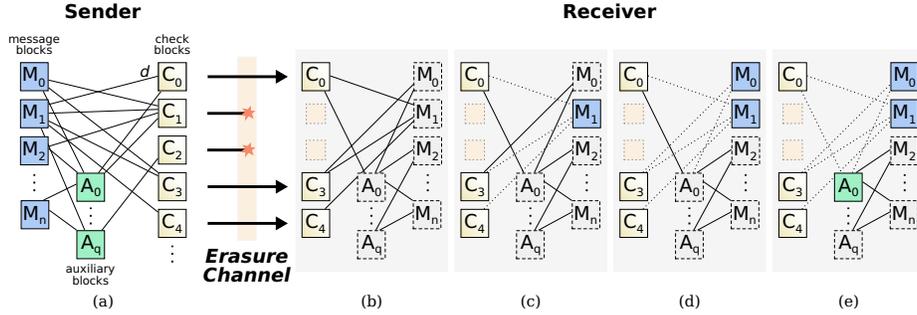
## 3 Reliable Transfer with Online-Coding

RTOC is a protocol for data transfer in networks that suffer from high and time-varying channel loss. The use of fixed, high-rate FEC schemes pay a constant but high overhead, and existing rateless approaches rely on end-to-end interactions, a fixed margin for loss tolerance, or incur relatively high decoding cost.

Application data (e.g., sensor data or program code) are assumed to be stored in pages or messages and are fragmented into blocks by RTOC for encoding and transmission to one or more neighboring nodes. After one roundtrip exchange to initiate the transaction, encoded data is streamed to the destination. Feedback control is used to determine when to slow and terminate transmission to minimize wasted communication, without requiring multiple rounds of ARQ or assuming that “no news is good news.” Before describing our solutions for synchronization and termination, we review online codes in Section 3.1.

### 3.1 Online codes

Online codes [8] are non-systematic fountain codes, developed independently from but similar to Raptor codes [11]. They concatenate two codes (outer and



**Fig. 1.** Sending a message  $M$  through an erasure channel with unbounded loss using online codes. Steps (b)–(e) show iterative belief propagation as blocks are received.

inner) to produce a limitless stream of output blocks from  $n$  original message or page blocks. Online codes improve on Luby’s LT codes [7] (used by SYNAPSE [6]) to achieve  $O(1)$  time encoding (per block) and  $O(n)$  decoding, trading optimal for near-optimal recovery performance. They are also locally encodable, which means that each output block is computed independently from the others, easing implementation and memory requirements on constrained WSN devices.

Encoding consists of an outer or pre-processing encoding followed by an inner encoding that generates output blocks, called “check” blocks. The outer encoding creates a fixed number  $q = k\delta n$  of “auxiliary” blocks that are appended to the original  $n$  message blocks ( $k$  and  $\delta$  are parameters described below). For every message block,  $k$  auxiliary blocks are chosen and the message block’s contents are exclusive-ORed with them all.

The inner encoding creates a potentially endless set of check blocks from the combined message and auxiliary blocks (the “composite” message). To generate a check block, a degree  $d$  is first chosen by sampling a distribution  $\rho$  with certain properties described below. Then  $d$  composite message blocks are chosen uniformly and exclusive-ORed together to make each check block. Construction of each check block is independent of all previous and future blocks (a property called local encodability), so it is easy to implement, takes only constant time, and requires little memory. Figure 1(a) shows message ( $M_i$ ), auxiliary ( $A_i$ ), and check blocks ( $C_i$ ) connected in a graph  $G$ , where edges from  $M$  to  $A$  represent the outer encoding and edges from  $M \cup A$  to  $C$  represent the inner encoding.

Check blocks are transmitted through the erasure channel to the receiver for recovery. To decode the page or message, the receiver uses a belief propagation algorithm on the subset of graph  $G$  formed by the blocks successfully received:

1. choose a check block  $C$  of degree one,
2. recover the contents of its adjacent block  $M_x$  as  $M_x = C \oplus M_i$ , for all  $M_i$  used to construct  $C$  originally and  $i \neq x$ ,
3. remove all edges in  $G$  of the recovered block  $M_x$ , and
4. repeat until the message is recovered or all check blocks have degree  $> 1$ .

Auxiliary blocks are decoded similarly, but after they have been recovered they are treated as check blocks and used to decode any remaining message blocks to which they are adjacent in  $G$ .

Figures 1(b)–(e) show an example of decoding steps at the receiver. Step (b) shows the state of graph  $G$  after check blocks  $C_0$ ,  $C_3$ , and  $C_4$  have been received. The first two blocks are buffered because their degrees are higher than one, but block  $C_4$  can be processed by the above algorithm upon its reception. In step (c), the contents of block  $C_4$  are copied to  $M_1$ , which is marked as recovered. Edges incident to  $M_1$  are deleted from graph  $G$ , so blocks  $C_0$  and  $C_3$  now have degree one. Block  $C_3$  is chosen next and used to recover block  $M_0 = C_3 \oplus M_1$  in step (d), and the edges of  $M_0$  are removed. Check block  $C_0$  similarly is used to recover auxiliary block  $A_0 = C_0 \oplus M_1$  in step (e), and edges between  $A_0$  and check blocks are removed. No more check blocks have degree one, so the algorithm terminates until another block is received.

Iterative decoding spreads the total processing cost across multiple block receptions, which is more friendly to co-hosted real-time processes than is batch decoding after  $k$  blocks are received as in Rateless Deluge.

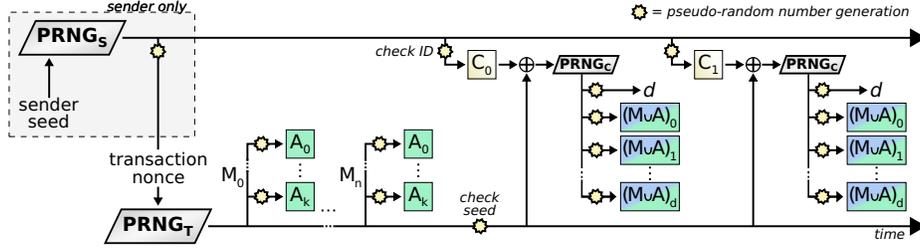
### 3.2 Synchronizing sender and receiver

Online and Raptor codes are in the family of fountain codes—so-called because they can generate endless streams of encoded blocks, and the receiver does not require any particular ones as long as a sufficient number of them are received. For unpredictable and arbitrarily low-capacity channels, this property allows RTOC to maintain communication. However, a mechanism is needed to shut off the flow of encoded blocks when the message has been recovered by the receiver, but without resorting to multiple rounds of control traffic.

To address these requirements, RTOC uses a lightweight protocol for synchronizing the parties, controlling the transmission rate, and terminating an exchange. We borrow the protocol nomenclature and sequencing from IEEE 802.11 messages, but redefine the semantics.

A *Request To Send* (RTS) message bears the transmitter’s block size  $b$ , total message length  $n$ , and a transactional nonce used to seed a pseudo-random number generator (PRNG). The message destination responds with a *Clear To Send* (CTS) message to acknowledge the RTS and indicate readiness to receive encoded fragments. Each encoded check block, or *DATA* message, bears the block’s identifier, which partially determines the random selections used to construct it. *DATA* blocks are streamed to the destination, and when the original message has been successfully decoded, the neighbor returns an *ACK* message and the exchange completes.

Senders and receivers must agree on the parameters of the online code,  $\epsilon$ ,  $\delta$ , and  $k$ , and on ephemeral or transactional state as well. The construction of graph  $G$  determines the composition of auxiliary and check blocks and must be synchronized. In particular, the random selection of message to auxiliary block mappings, and the generation of random samples from distribution  $\rho$  for constructing check blocks must be performed identically by both parties.



**Fig. 2.** Senders and receivers use synchronized transaction state to compute loss-tolerant pseudo-random block mappings for the online code.

The sequence and dependencies of these steps are shown in Figure 2, viewing from left to right. The sender of the application message seeds  $\text{PRNG}_S$  with a private value not shared with the receiver, though it need not be secret. This PRNG is later used to generate a random identifier for each check block that is sent. After exchange of the RTS and CTS messages, both parties generate the subset of  $G$  consisting of the  $kn$  edges defining message block to auxiliary block mappings, using the *transaction nonce* to seed  $\text{PRNG}_T$ . A final value, the *check seed*, is generated from this PRNG to combine with each check block identifier.

Our solution of combining multiple generators satisfies several objectives. Sequentially numbering check blocks would produce a highly autocorrelated input for generating the check block contents, resulting in poor randomness when used with a linear-feedback shift register. For this reason the identifiers are randomly generated by the sender. However, they may be too short to produce a long-period sequence because of constraints on identifier length. They are therefore combined with the check seed and used to seed  $\text{PRNG}_C$  for generating the check block degree  $d$  from  $\rho$  and the adjacent message and auxiliary blocks. As the check seed is derived from the transaction nonce, it also provides randomness among multiple messages sent by a single node.

Receivers must be able to determine the contents of each check block independently to cope with loss. This is satisfied by using the identifier, which is unique to the received block, together with the check seed, which is unchanging for all blocks created from a single application page or message. Separately seeding  $\text{PRNG}_C$  with this combination ensures that both endpoints produce the same pseudo-random stream: first the degree  $d$ , then  $d$  blocks to exclusive-OR together (at the sender) or mark as adjacent and decode (at the receiver).

Hence, after a single round-trip exchange to begin the transfer, data flows until an ACK stops it. These are retransmitted if necessary to overcome loss, but no other control messages are needed.

### 3.3 Stream termination and rate control

Special attention must be given to terminating the stream of check blocks, which is potentially endless. When channel capacity is low, the effective rate ( $n/c$ , where

$c$  check blocks were transmitted in total) necessary to recover the data may be quite low. Rather than fix the number of check blocks to transmit, which assumes accurate knowledge of the loss rate or requires additional control messages to finish, we use online rate control. This makes our protocol more robust to high loss rates of both data and control messages and to dynamic channel conditions.

Every node maintains an estimate of the loss rate  $\hat{\gamma}$  for each neighbor link to determine how many check blocks to transmit. When an application message is successfully transmitted to a neighbor at time  $t$ , the sender saves the number of check blocks  $c_t$  that were required (as reported by the receiver in the terminating ACK message) and total message blocks  $n$ . Using the expectation from [12] that messages are recoverable with high probability from  $(1 + 3\epsilon)n$  check blocks, the average loss rate for the completed transmission is computed as:

$$\gamma_t = 1 - \frac{(1 + 3\epsilon)n}{c_t} \quad (1)$$

The estimated current channel loss to the neighbor is updated as an exponentially weighted moving average:

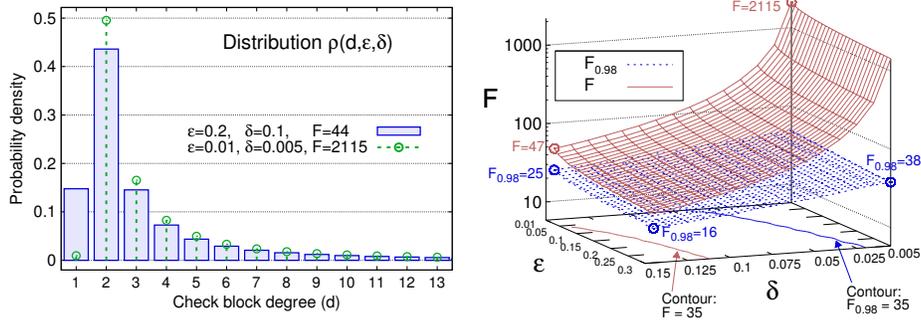
$$\hat{\gamma}_{t+1} = \alpha\gamma_t + (1 - \alpha)\hat{\gamma}_t \quad \text{for } 0 \leq \alpha \leq 1 \quad (2)$$

We estimate the channel loss instead of  $c$  directly because we allow the length  $n$  to vary freely among application pages or messages.

When sending a message at time  $t + 1$  to the same neighbor, the node transmits  $(1 + 3\epsilon)n / (1 - \hat{\gamma}_{t+1})$  check blocks at the nominal rate supported by the underlying MAC layer. If no ACK has been received to terminate the transmission by this time, the node reduces the sending rate, but continues to send check blocks up to some maximum tolerated  $c_{\max}$ . The lower but sustained rate reduces overhead at the sender, while allowing for potentially high losses on the reverse link that interfere with ACKs.

Prior to transaction termination with an ACK message, the receiver may periodically notify the sender of the number of check blocks received and message blocks decoded. The sender then updates  $\hat{\gamma}$  to shorten or extend the duration of the full transmission rate period. However, as channel losses may be severe, the original  $\hat{\gamma}$  is used if no updates are received.

In contrast with ARQ protocols, which implode under retransmissions in lossy networks [4], RTOC's transaction control mechanism tolerates high losses. Given that the original RTS and CTS messages are repeated sufficiently many times to overcome channel loss  $\gamma$  and begin the online coding, the receiver can recover the message from any  $(1 + 3\epsilon)n$  check blocks. No further acknowledgment is required, though it does prevent the sender from wasting transmissions up to the maximum tolerated  $c_{\max}$  (equivalently, down to a minimum effective rate  $n/c_{\max}$ ). High loss conditions that would prevent acknowledgment delivery are also when the maximum number of check blocks are likely to be required, so the waste is small. Conversely, in good conditions when the potential waste  $c_{\max} - (1 + 3\epsilon)n$  is high, an ACK terminates the transaction promptly.



(a) Probability density of  $\rho$ , with high density in low degrees and a long, thin tail. Small  $\epsilon$  produce very small  $\rho_1$  densities. (b)  $F$  and 98% bound  $F_{0.98}$  for  $\rho$ . Contours show feasible region for  $n + q = 35$ .

**Fig. 3.** Impact of parameters  $\epsilon$  and  $\delta$  on the check block degree distribution  $\rho$  and maximum degree  $F$ , computed from Equation 3.

## 4 Design of Code Parameters

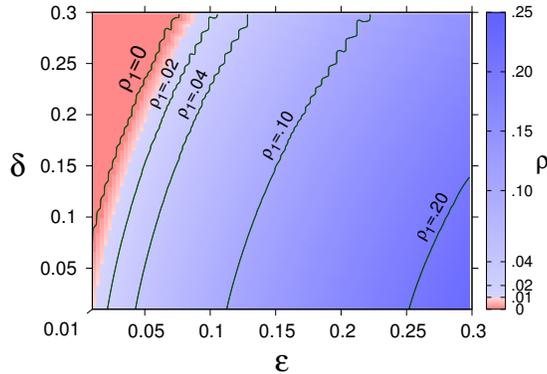
A key challenge remaining for the use of online codes is the selection of several inter-related encoding parameters that determine efficiency and suitability for use in WSNs. Maymounkov's analysis of the coding scheme and degree distribution  $\rho$  shows that the receiver can recover the original data with high probability after receiving  $(1 + 3\epsilon)n$  check blocks [8]. Distribution  $\rho(d, \epsilon, \delta)$  is given as:

$$\rho_1 = \frac{1 - 1/F}{1 + \epsilon}, \quad \rho_i = \frac{(1 - \rho_1)F}{(F - 1)i(i - 1)} \quad \text{for } 2 \leq i \leq F, \quad F = \left\lceil \frac{\ln(\epsilon/2) + \ln \delta}{\ln(1 - \delta)} \right\rceil \quad (3)$$

A small  $\epsilon$  minimizes transmission overhead, however, it also skews distribution  $\rho$  to the right. This increases the average check block degree  $d$  and the decoding complexity which is proportional to  $n \ln(\epsilon/2)$ . Figure 3(a) shows the probability density of  $\rho$  for  $\epsilon = 0.2$  and  $\epsilon = 0.01$ , with  $\delta = \epsilon/2$  in both cases. For  $\epsilon = 0.01$ , the value recommended by Maymounkov, the maximum check block degree  $F$  given by Equation 3 is 2115—far exceeding the number of composite message blocks  $n + q$  needed in this context and consuming valuable memory space. The lookup table for sampling  $\rho$  requires up to 4230 B, which is more than the capacity of MicaZ's SRAM.

However, we note that  $\rho$  has a long, thin tail with more than half of its density concentrated in its first two elements  $\rho_1$  and  $\rho_2$ . An implementation could truncate the distribution with little practical effect on decoding performance if large values sampled from  $\rho$  are very rare. To gain a better understanding of the usable range of  $\epsilon$  and  $\delta$  parameters given WSN constraints, we numerically calculated the least degree  $d$  that bounds 98 percent of the cumulative probability density of distribution  $\rho$ :  $F_{0.98} = \min d$ , such that  $\sum_{i \leq d} \rho_i \geq 0.98$ .

Figure 3(b) shows  $F_{0.98}$  for values of  $\epsilon \in [0.01, 0.3]$  and  $\delta \in [0.005, 0.15]$ , and indicates that small values of  $\epsilon$  may be practical depending on the number of



**Fig. 4.** Contour map showing  $\rho_1$  for  $\epsilon, \delta \in [0.01:0.3]$ . Parameter settings in the upper-left region do not produce check blocks with degree one, which are required by the given decoding algorithm.

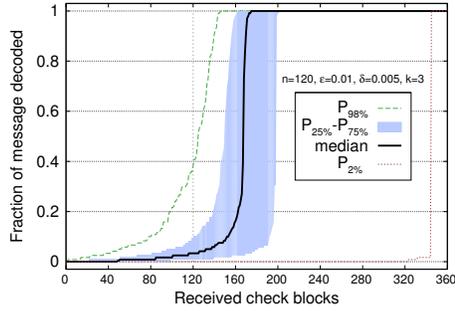
blocks  $n$  created by fragmentation. For example, a 480 B page sent as 16 B blocks creates about 35 composite blocks to be selected randomly from the distribution of  $\rho$ . Values of  $\epsilon$  and  $\delta$  for which  $F$  and  $F_{0.98} \leq 35$  are indicated in Figure 3(b) by contour lines. Truncating  $F$  makes a much larger parameter space available, as seen from the difference in the contours.

Parameters  $k$  and  $\delta$  affect the online code performance in three ways. First, the number of auxiliary blocks created for each message grows as  $q = k\delta n$ , so for fixed  $k$ , higher  $\delta$  values require more memory buffers. Second, higher  $\delta$  values increase the probability  $\delta^k$  that the message cannot be decoded until after  $(1+\epsilon)n$  blocks. Although the foregoing argue against large  $\delta$ , the last consideration is that small  $\delta$  values, as for small  $\epsilon$  values, extend the right-tail of  $\rho$  and increase decoding complexity.

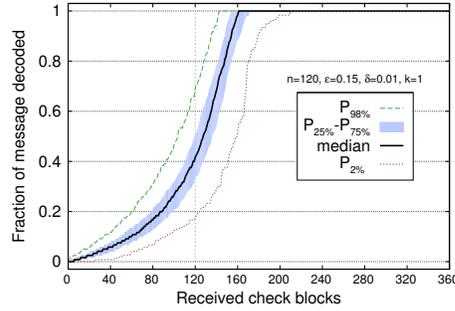
There is one further disadvantage of a small  $\epsilon$ —and which is ultimately determinant—that affects the use of an online code in our setting. Using the algorithm given above, recovery of the message cannot begin until a check block of degree one (i.e., a copy of a message block) is received. With relatively few message blocks to send in total, it may often happen that the first check block to be randomly assigned degree one is sent very late, requiring the receiver to buffer check blocks well in excess of the asymptotically expected  $(1 + 3\epsilon)n$  bound.

The probability  $\rho_1 = \rho(d=1, \epsilon=0.01, \delta=0.005)$  of such a degree-one block is only 0.0094, as shown in Figure 3(a). For this low  $\rho_1$  density, there is a 30% chance that a check block of degree one is not sent until after 127 others, which delays decoding and increases buffer occupancy at the receiver.

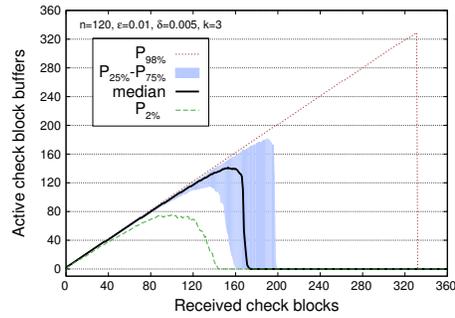
Maymounkov and Mazières [12] make the simplifying recommendation that  $\delta$  be chosen as  $\epsilon/2$ . However, subject to the constraints and trade-offs discussed,  $\epsilon$  and  $\delta$  may be varied independently. A contour map of  $\rho_1$  for values of  $\epsilon$  and  $\delta$  is shown in Figure 4. The upper-left region is infeasible for the given algorithm



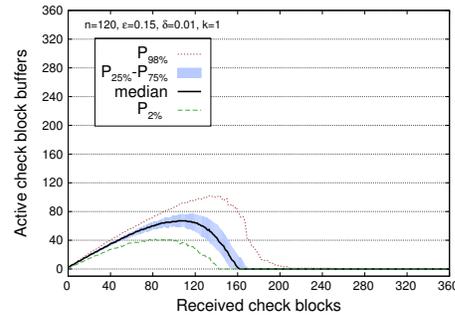
(a) Message decoding progress for a 120-block message as check blocks are received. Small  $\epsilon = 0.01$ ,  $\delta = 0.005$  cause high variance in this region.



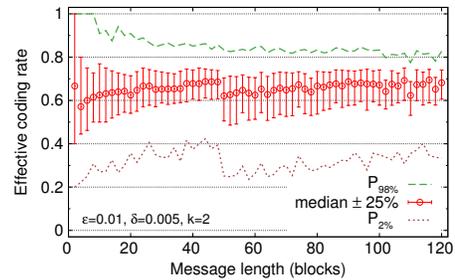
(b) The full 120-block message is decoded earlier on average for larger  $\epsilon = 0.15$ ,  $\delta = 0.01$ , incurring less transmission overhead.



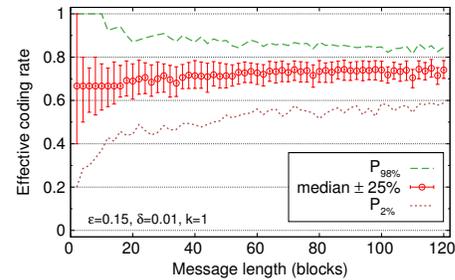
(c) Number of active check block buffers for simulation shown in (a) above. For 2% of the tests, more than 320 buffers were required.



(d) Significantly fewer active buffers were required by the simulation shown in (b) above, for more predictable memory management and buffer re-use.



(e) As the message length  $n$  increases, the effective coding rate exhibits high variance due to a low  $\rho_1$  density from the small valued parameters.



(f) Larger  $\epsilon$  increases the  $\rho_1$  density, giving a more consistent and overall higher effective coding rate in this region.

**Fig. 5.** Simulation results show the influence of  $\epsilon$  and  $\delta$  on cumulative decoding progress, active check block buffers, and the effective coding rate. The median, and 2<sup>nd</sup>, 25<sup>th</sup>, 75<sup>th</sup> and 98<sup>th</sup> percentiles of 500 runs are shown.

because  $\rho_1$  is either zero or very small, and must be avoided. Outside this region, the parameters may be chosen to yield good performance, as we now describe.

Rather than optimizing for asymptotic behavior, selection of higher  $\epsilon$  values leads to better performance in the operating region useful for WSNs, that is, for relatively small  $n$ . We implemented the encoding and decoding algorithm on a PC to measure the code performance trade-offs for the parameters  $\epsilon$ ,  $\delta$ , and  $k$ . From the data collected, we present the median, and 2<sup>nd</sup>, 25<sup>th</sup>, 75<sup>th</sup> and 98<sup>th</sup> percentiles of 500 randomized runs per data point.

Figures 5(a) and 5(b) show the significant differences in the cumulative decoding progress for  $\epsilon = 0.01$  and  $\epsilon = 0.15$ , respectively, when receiving the check blocks transmitted for a message of  $n = 120$  blocks. A small  $\epsilon$  yields a small  $\rho_1$  and delays the decoding, as evidenced by the larger variance and extended recovery time shown in Figure 5(a). In 2% of the tests, very little of the message was recovered until after 340 check blocks ( $\approx 2.8n$ ) had been received, when decoding rapidly proceeded.

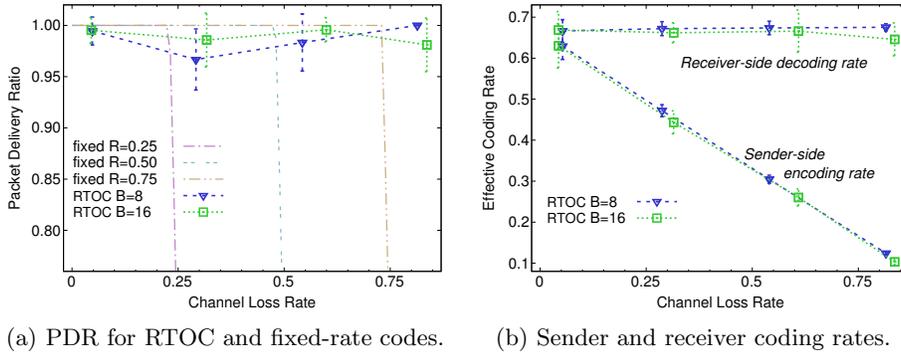
A larger  $\epsilon = 0.15$  parameter yields significantly more compact and consistent performance and the final decoding time is faster, as shown in Figure 5(b). This is particularly beneficial on memory-constrained WSN devices, as buffers allocated to check blocks can be re-used when a block is decoded completely. Figures 5(c) and 5(d) show the large difference in dynamic check block buffer use, with 52% (vs. median) and 69% (vs. 98<sup>th</sup> percentile) less memory required than for  $\epsilon = 0.01$ . Fixed memory overhead from the  $k\delta n$  auxiliary blocks is also kept low by parameters  $\delta = 0.01$  and  $k = 1$ .

The effective coding rate (i.e., ratio of the number of message blocks to check blocks) was measured for  $2 \leq n \leq 120$ , and indicates that these effects are even more pronounced for smaller message lengths. Figure 5(e) shows the wide variability in effective rate for lengths smaller than the  $n = 120$  case of Figures 5(a)–5(d). A linear fit of the  $\epsilon = 0.15$  data shown in Figure 5(f) gives an effective rate of 0.7–0.75 with smaller variance.

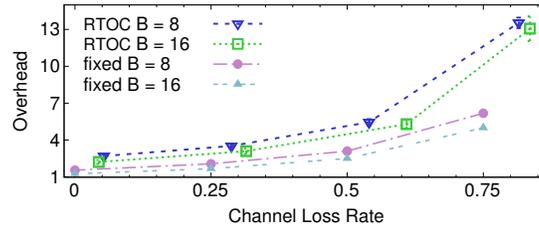
Our analysis of the online code degree distribution  $\rho$ , the impact of its key parameters, and simulation results in the domain of WSN operation lead to the selection of a higher  $\epsilon$  to reduce variance by 50–71% and a relatively small  $k$  and  $\delta$  to reduce fixed overhead. This enables implementation on memory-constrained devices, and trades asymptotic efficiency for good performance in RTOC, and allows the transport protocol to benefit from online coding’s algorithmic advantage over other proposed rateless schemes.

## 5 Experimental Results

Having analyzed and optimized the online coding parameters, we evaluated the performance RTOC in an embedded implementation for the MicaZ mote to obtain the most realistic and accurate results possible. First, we consider the effect of channel loss on link reliability and transmission overhead. We utilize a loss, or erasure, rate of  $\gamma$  on all protocol messages and fragments.



**Fig. 6.** Effect of channel erasures on packet delivery ratio and effective coding rates, using parameters  $\epsilon = 0.15$ ,  $\delta = 0.01$ ,  $k = 1$ .



**Fig. 7.** Message transmission overhead (ratio of bytes transmitted, including headers, to bytes of application payload) rises significantly for high loss rates. Shown for comparison are fixed-rate codes.

In the following tests, a node transmitted 180 messages of length 96 B to a neighbor using block (fragment) lengths of 8 and 16 bytes, and we measured the performance under varying loss rates. We induced nominal loss rates of 0–75% by discarding blocks randomly in software on the MicaZ motes. Uncontrolled channel conditions at the time of the experiments further lowered the actual loss rates, which are shown in the figures.

Despite actual channel plus induced loss rates of up to 84%, RTOC delivers in excess of 95% of packets. Figure 6(a) shows the mean and standard deviation of PDRs for the tests, which range from 95–100%. Induced losses were applied equally to data fragments and transaction control messages, which must be retransmitted to keep the protocol from stalling.

Online codes are designed to decode a message with high probability after  $(1 + \epsilon)n$  check blocks are *received*—which clearly requires many transmissions to overcome high loss rates. Figure 6(b) shows the effective coding rates for both sender and receiver. The receiver’s rate is the ratio of the total number of message blocks to the number of check blocks *received before decoding success*. It is nearly constant in the range of 0.64 to 0.69 and is consistent with the

simulation results presented earlier in Section 4. The sender’s encoding rate is the ratio of the total number of message blocks to the number of check blocks *sent before protocol termination*, and directly reflects the increasing channel loss, as it drops from 0.63 to about 0.12.

Figure 7 shows the overhead of the RTOC protocol, fragmentation, and channel losses more directly. We measured the overhead as the ratio of all bytes transmitted, including fragment and TinyOS headers, for the original 96 B payload length. High loss rates, as expected, require the most transmissions and incur high overhead. Larger block sizes are more efficient because the overhead from headers is greatly reduced.

For comparison, the behavior of ideal fixed-rate error correction schemes for rates 0.25, 0.5, and 0.75 is also shown in Figures 6(a) and 7. Fixed-rate codes enjoy small overheads, which are calculated from the design rate (1 to 0.25), header length (5 B), and block size (8–16 B), when matched to the actual loss rate. However, as these schemes are designed to correct only a fixed fraction of errors, PDR drops precipitously when the loss rate exceeds their design rate.

The overheads of RTOC and fixed-rate codes are the result of both the coding rate and fragmentation. Fragmentation alone incurs substantial overhead depending on the block size. For original message payload length  $P$ , header length  $H$ , block size  $B$ , fixed coding rate  $R$  or number of check blocks transmitted  $c$ , the overheads are:

$$\text{Fixed-rate} = \frac{\lceil \frac{P}{B \cdot R} \rceil \cdot (H + B)}{H + P}, \quad \text{Online code} = \frac{c \cdot (H + B)}{H + P} \quad (4)$$

Short fragments give high overheads, but may be necessary due to application constraints, and may be less prone to erasure in very poor channels.

## 5.1 Discussion

Many systems for WSNs must be adaptable at runtime to handle the wide performance range between normal operation and when channels are very poor. The overhead of RTOC is primarily due to: (1) fragmentation of pages or messages into smaller blocks, (2) message expansion from the effective coding rate, and (3) streaming of fragments in a transaction. RTOC allows trade-offs in these key areas to maintain efficiency and incur overhead only when necessary for loss resistance. Most mechanisms are automatic and part of the design of protocol, while selection of the block size is exposed as part of its configuration to allow external control by the application.

Use of a rateless erasure code overcomes variable channel loss rates automatically with proper parameter selection and integration with the transaction control protocol. Through analysis of online coding’s degree distribution, we chose parameters  $\epsilon$ ,  $\delta$ , and  $k$  to achieve stability and good performance in the operating region useful for WSNs. The resulting low coding rate variance reduces memory pressure on already constrained WSN devices.

The check block transmit rate control algorithm described uses the estimated loss rate  $\hat{\gamma}$  and bound  $c_{\max}$  on check blocks to reduce wasted transmissions

from lost termination (ACK) messages. These mechanisms automatically adjust RTOC's behavior to prioritize message delivery despite poor channel conditions. In our embedded evaluation, the protocol transferred over 95% of the messages successfully despite up to 84% induced channel loss.

## 6 Conclusion

Despite the resource limitations of WSN devices and high channel loss, online coding and RTOC's synchronization and termination mechanisms provide efficient, reliable data transfer that can serve as a building block for data exfiltration or code updating. We carefully designed the protocol's parameters to trade asymptotic optimality for predictability in the WSN operating region, and therefore it imposes modest memory and resource requirements on the system. We presented an evaluation of its implementation on embedded hardware to demonstrate its efficiency and performance. Future work may apply our methods to other codes, such as Raptor codes [11], and integrate RTOC with over-the-air reprogramming protocols for high-loss networks.

## References

1. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proc. of SenSys, Baltimore, MD (2004) 81–94
2. Paek, J., Govindan, R.: RCRT: rate-controlled reliable transport for wireless sensor networks. In: Proc. of SenSys, Sydney, Australia (2007) 305–319
3. Kim, S., Fonseca, R., Dutta, P., Tavakoli, A., Culler, D., Levis, P., Shenker, S., Stoica, I.: Flush: a reliable bulk transport protocol for multihop wireless networks. In: Proc. of SenSys, Sydney, Australia (2007) 351–365
4. Kumar, A.: Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM TON* **6**(4) (1998) 485–498
5. Hagedorn, A., Starobinski, D., Trachtenberg, A.: Rateless Deluge: Over-the-air programming of wireless sensor networks using random linear codes. In: Proc. of IPSN, St. Louis, MO (2008) 457–466
6. Rossi, M., Zanca, G., Stabellini, L., Crepaldi, R., Harris, A., Zorzi, M.: SYNAPSE: A network reprogramming protocol for wireless sensor networks using fountain codes. In: Proc. of SECON, San Francisco, CA (2008) 188–196
7. Luby, M.: LT codes. In: Proc. of IEEE Symposium on Foundations of Computer Science. (2002) 271–280
8. Maymounkov, P.: Online codes. Technical report, New York University Technical Report (2002)
9. Heidemann, J., Silva, F., Estrin, D.: Matching data dissemination algorithms to application requirements. In: Proc. of SenSys, Los Angeles, CA (2003) 218–229
10. Levis, P., Patel, N., Culler, D., Shenker, S.: Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: Proc. of NSDI, San Francisco, CA (2004) 15–28
11. Shokrollahi, A.: Raptor codes. *IEEE Transactions on Information Theory* **52** (2006) 2551–2567
12. Maymounkov, P., Mazières, D.: Rateless codes and big downloads. In: Proc. of IPTPS, Berkeley, CA (2003)