2008 International Conference on Information Processing in Sensor Networks

Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks using Random Linear Codes

Andrew Hagedorn, David Starobinski, and Ari Trachtenberg
Dept. of Electrical and Computer Engineering
Boston University, Boston, MA 02215
Email: {achag,staro,trachten}@bu.edu

***Abstract*— Over-the-air programming (OAP) is a fundamental service in sensor networks that relies upon reliable broadcast for efficient dissemination. As such, existing OAP protocols become decidedly inefficient (with respect to energy, communication or delay) in unreliable broadcast environments, such as those with relatively high node density or noise. In this paper, we consider OAP approaches based on rateless codes, which significantly improve OAP in such environments by drastically reducing the need for packet rebroadcasting. We thus design and implement two rateless OAP protocols, rateless Deluge and ACKless Deluge, both of which replace the data transfer mechanism of the established OAP Deluge protocol with rateless analogs. Experiments with Tmote Sky motes on single-hop networks with packet loss rates of 7% show these protocols to save significantly in communication over regular Deluge (roughly 15-30% savings in the data plane, and 50-80% in the control plane), and multi-hop experiments reveal similar trends. Simulations further shows that our new protocols scale better than standard Deluge (in terms of communication and energy) to high network density. TinyOS code for our implementation can be found at http://nislab.bu.edu.**

I. Introduction

Sensor networks distinguish themselves in their unique capability of gathering detailed information in remote, isolated, and often harsh environments. Yet, sensor networks' software often needs to be updated after deployment for a variety of reasons, such as fixing software bugs, modifying tasks of individual nodes or of the entire network, and patching security holes. Within this context, over-the-air programming (OAP) protocols play a key role as an enabling technology to numerous sensor network applications, and several protocols and algorithms have been specifically designed for this purpose.

Although existing OAP protocols have many merits, they suffer from fundamental limitations that can significantly impair their use in future systems. Chiefly, the performance of existing OAP protocols quickly degrades when the network size and density get large, and even more so when packet loss is high. The survey work in [12] reports simulation results, based on TOSSIM, where the completion time of Deluge [2] and MNP [4], two popular OAP protocols, can easily take close to an hour on a 100-node network. This lack of scalability can largely be attributed to the high control-plane overhead associated with reliability requirements, and most specifically with negative acknowledgment (NACK) mechanisms. These mechanisms require every destination to contend on a shared channel to notify the source about its missing packets, producing the so-called "NACK implosion problem".

Our main contribution in this work is to devise and fully implement a fundamental solution to the aforementioned scalability challenges faced by OAP. Our approach relies on the use of rateless coding to eliminate the need to convey control information about *which* packets require retransmission; with this approach, a node need only receive a sufficient number of distinct, encoded packets to recover a transmitted program. Implementing rateless codes in the resource constrained environment of a wireless sensor requires the design of efficient mechanisms to reduce latency, computational complexity, and memory overhead. To demonstrate the effectiveness of using rateless codes for OAP, we propose and implement two new protocols. The first protocol, called *rateless Deluge*, significantly alters the transfer mechanism of the OAP Deluge protocol to allow for the rateless transfer of program images. The second protocol, called *ACKless Deluge*, augments the rateless Deluge protocol with a packet level forward erasure correction (FEC) mechanism that aims at eliminating the need for extraneous control packets. ACKless Deluge transmits extra encoded packets that prevent, with high probability, the need for packet retransmissions.

In this paper, we provide a detailed description of the implementation of these two new rateless-based OAP protocols. In particular, we shed light on the various trade-offs that arise in implementation of rateless OAP on a sensor networks, such as the tradeoff between the size of program pages and the size of the underlying finite field used for computation. We provide extensive numerical results evaluating the performance of our protocols, based both on real network experiments with Tmote Sky sensors and also on simulations. We show that precoding, whereby new packets are encoded in anticipation of future requests, can be exploited to substantially speed-up the data transfer mechanisms of the rateless protocols. Our results further indicate that the new protocols achieve significant savings of energy and communication with respect to the standard version of Deluge (over 50% in many cases). Their overall completion times are comparable in low network densities or low packet loss environments, but better than original Deluge as packet loss rate or network density increases.

978-0-7695-3157-1/08 $25.00 © 2008 IEEE
DOI 10.1109/IPSN.2008.9

A. Paper organization

The rest of this paper is organized as follows. We first briefly review state-the-of-art OAP protocols and provide background on random linear codes in Sections II and III. Next, in Section IV, we describe the implementation of the rateless OAP protocols, with a special focus on the memory and computational overheads that they entail. We also describe the design of our FEC technique that reduces the effects of packet loss and prevents requests for re-transmission. In Section VI we compare the efficiency of the original Deluge to the rateless implementations through experiments on our testbed and simulations. Our conclusions are presented in Section VIII.

II. Related Work

We survey here work directly related to OAP protocols. A number of such protocols have been proposed in the last few years. Among them, the Deluge protocol [1, 2] is currently the *de facto* standard. In Deluge, each node periodically advertises the most recent version of its program, and nodes request (and receive) program updates based on these advertisements using a NACK-based protocol for reliability. In order to reduce contention on the shared channel, Deluge implements *advertisement and NACK suppression*, which aim at avoiding redundant transmissions of control packets. In addition, to enable *pipelining*, a program is divided into fixed-size segments (or pages), which in turn are divided into packets. As soon as a node receives an entire segment, it can forward it on to its neighbors. In [1], forward error correction in the form of Reed Solomon codes and Tornado Codes is proposed as a potential optimization of the Deluge protocol, but these codes have a fixed rate and therefore require the retransmission of entire pages if too many packets are lost.

OAP protocols that preceded Deluge include XNP, used in TinyOS for single-hop reprogramming, and Multi-hop Over-the-Air Programming (MOAP) [11]. MOAP is similar to Deluge, but does not divide a program into pages. More recent protocols include Multi-hop Network Programming(MNP)[4], Infuse [3] and Sprinkler [7]. MNP implements sender selection to limit the number of concurrent transmissions in each neighborhood. Both Infuse and Sprinkler propose to set up a TDMA schedule to reduce packet collisions. A detailed description and comparison of these methods and a few others is provided in [12].

While our implementations are based on the Deluge protocol, we expect that the new rateless coding transfer mechanisms described in this paper could be overlayed on any of the above protocols to substantially improve their performance.

Significant efforts have recently been devoted in developing new macro-programming methods and middleware to reduce the amount of data needed to update and modify programs; see e.g., [5, 6, 8]. This paper's contribution should be viewed as complementary to these efforts.

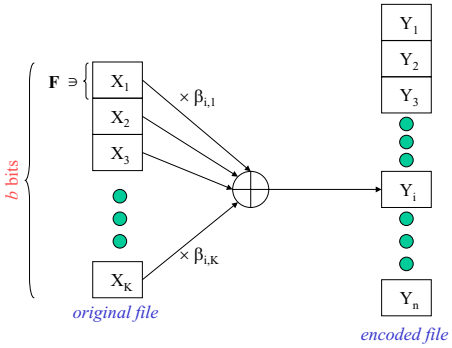

Fig. 1. Illustration of random linear encoding

III. Background on Rateless and Random Linear Codes

Rateless codes provide an efficient means of addressing channel contention in sensor networks, while at the same time minimizing control messages, such as those contributing to the ACK/NACK implosion problem. Fundamental to this strategy is the fact that receivers do not need to indicate which specific packets require retransmission; instead, they just have to receive a sufficient number of independent packets, which can then be used to decode the original message. Rateless coding, thus, yields several key benefits, namely: communication and energy savings, and lower control overhead.

Random linear coding provides a simple method for file dissemination. In this model, a long file X is split into k segments $X_1, X_2, \ldots X_k$, each of which can be thought of as an element in a finite field $\mathbb{F}$. These segments are then encoded into $m > k$ messages $\{Y_1, Y_2, \ldots Y_m\}$ as the following random linear combinations $Y_i = \sum_{j=1}^{k} \beta_{i,j} X_j$, where $\beta_{i,j}$ are randomly chosen elements in the finite field $\mathbb{F}$. The parameters $\beta_{i,j}$ of the encoding can be easily adjusted so that the rows $[\beta_{i,1}, \beta_{i,2}, \ldots \beta_{i,k}]$ are linearly independent with high probability. Thus, any host that receives k of the Y_i's can solve the corresponding system of linear equations to determine X. Figure 1 graphically demonstrates the encoding process.

This technique sports two features useful to OAP: (i) it has no decoding inefficiency; and (ii) it is a *rateless* code. In classical block codes, the encoding length needs to be determined *a priori*. In this case, however, if the m encodings $Y_1 \ldots Y_m$ prove insufficient (due to poor channel conditions, for example), then the encoding node can easily generate a number of extra packets Y_i by using newly constructed random elements β.

A. Motivating Example

As a simple example of the best-case gain achievable with rateless codes, consider a one-hop clique consisting of a base

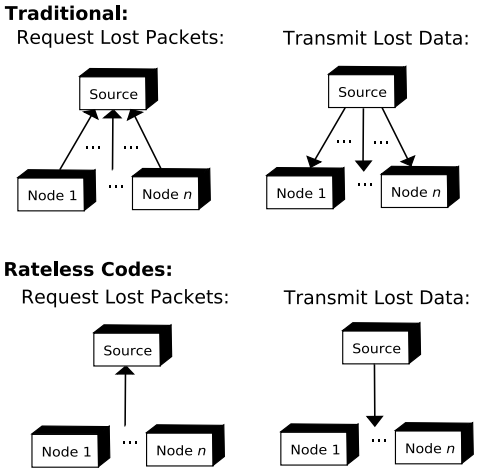

Fig. 2. A motivating example of the best-case gain achievable with rateless codes. If each receiver misses a different packet, the traditional mode requires the tranmission of n requests and n retransmissions. However, with rateless codes only 1 request and 1 retransmission is required.

station and n sensor nodes. Suppose that the base station broadcasts n data packets and each node in the network fails to receive a different packet. A traditional data dissemination protocol, such as Deluge, would require that each sensor transmits (on one shared broadcast channel) a NACK control packet with the ID of its missing packet. Upon reception of these n NACKs, the base station would have to retransmit all n data packets.

On the other hand, with rateless coding only one sensor needs to request the transmission of an additional encoded packet (assuming the other nodes can overhear that request). Once the base station transmits one new packet, each node can use this packet to recover the data. Rateless coding thus yields an n-fold reduction in communication cost on *both* the control and data planes in this case.

IV. Implementation of Rateless OAP

The implementation of a rateless OAP involves two crucial elements. The first element is the design of the rateless code to minimize latency, computational complexity, communication, and energy use. In general, these design choices are influenced by underlying hardware restrictions, such as the amount of memory available and the processor architecture and speed. The second element involves re-engineering the OAP data transfer mechanism. The new rateless data transfer mechanism must naturally integrate with the existing OAP protocol. This portion of the design is highly specific to the OAP selected (in our case, Deluge).

A. Finite Fields

The selection of the finite field size q of field F_q has a considerable impact on the performance of the system in terms of the computational complexity and probability to decode successfully. Decoding fails when the k rows of the randomly chosen $k \times k$ matrix are linearly dependent. The probability of decoding failure is based on well-known considerations of the number of linearly independent $k \times k$ matrices:

$$\Pr(\text{failure}) = 1 - \frac{(q^k - 1)\Pi_{i=1}^{k-1}(q^k - q^i - 1)}{q^{k^2}}$$

Increasing the field size q also increases probability of proper decoding at the expense of increased computational complexity for finite field arithmetic. For any reasonable field size, performing arithmetic on the fly is computationally demanding, since multiplication is performed modulo an irreducible polynomial and division requires an application of the extended Euclidean algorithm [9][1]. This computation complexity can be traded off for memory by precomputing multiplication or inverse tables. We chose a field of size $q = 2^8$ (corresponding to byte-length elements), which requires 256 bytes of memory to store inverses and has a probability of decoding failure of ~ 0.00392. By way of comparison, the next byte-aligned field size of $q = 2^{16}$ requires a $65KB$ table, which surpasses the 10KB memory on the Tmote Sky motes.

B. Random Linear Codes

Our implementation of random linear codes is divided into two parts: encoding and decoding. During encoding, a random number generator is seeded with a key shared by all nodes and a unique packet identifier to create random coefficients for encoding a given packet (if security is a requirement, then the shared key should be kept secret and distributed using a secure key distribution scheme). Once the current page is encoded into a packet, that packet and its identifier are transmitted over the channel. Including the identifier in each transmission allows the decoding mote to recreate the row of the random matrix used to encode the data packet; the identifier and key are combined to form a seed for the random number generator. Once all rows of the matrix have been generated, the decoding process uses Gaussian elimination with back substitution to solve the set of linear equations and retrieve the data. If decoding fails, the process recovers gracefully by only discarding those packets that are linearly dependent (indicated by zero rows in the reduced decoding matrix). The decoding mote must retrieve enough new packets to replace the dependent packets, and then it can generate the corresponding new rows of the random matrix. The node repeats the decoding process until it has obtained linearly independent packets and decoding succeeds.

The performance of random linear coding depends on the number and size of the packets being decoded (i.e., the size of the matrix). Both of these values are constrained by the resources available to the motes: the default maximum data payload size for TinyOS is 29 bytes and there is a fixed amount of RAM. The default value for Deluge is a 48 packet page, where each packet contains 23 bytes of data. This data payload size represents a worst case in terms of computational

[1]There are more efficient approaches to finite field arithmetic, such as using special bases or picking trinomial irreducibles, but the fundamental issues remain.

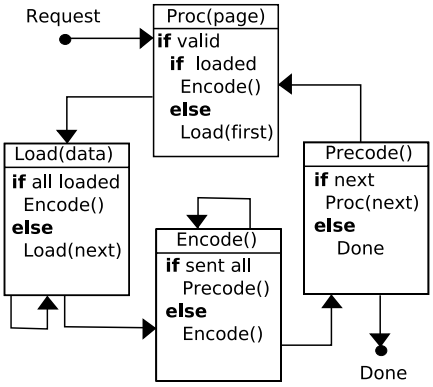

Fig. 3. State diagram at the source for a valid request for a data packet. Upon reception of the request the source loads all data, encodes, and transmits encoded packets. After transmitting the required number of encoded packets the source precodes the next page if available.

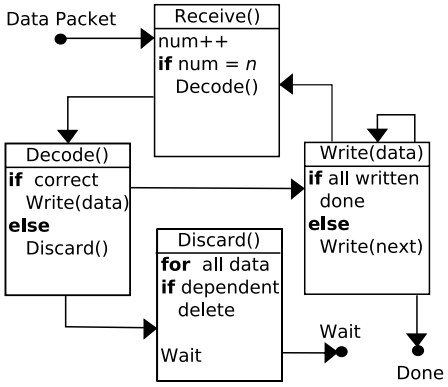

Fig. 4. State diagram at the receiving node for a valid data packet. Once the node has received k encoded packets it attempts to decode. If decode is successful the node writes the data to Flash. Otherwise the node discards any linearly independent packets and waits for more encoded packets.

complexity for our algorithm; if the payload is smaller and the number of packets per page remains the same, both encoding and decoding will take less time. Due to the use of Gaussian elimination, the decoding time [2] is $O(k^3)$, where the size of the random matrix is $k \times k$. This means that the page size should be kept small to reduce computational complexity. For example, on Tmote Sky motes requires 6.96 seconds, on average, to decode a 48 packet page. Reducing the page size to 24 packets per page decreases the average decoding time to 1.96 seconds.

C. Rateless Deluge

We next describe the full implementation of our first OAP protocol called rateless Deluge. Rateless Deluge modifies the original Deluge protocol in that it uses rateless codes to transmit data. This change causes significant structural changes to the mechanism for requesting and transferring data so that communication in the control and data planes are reduced. To ensure a fair comparison, all the other aspects of original Deluge such as image advertisement and data storage are kept identical. Hence, the only difference between the two implementations lies in their transfer mechanisms.

The change to the request mechanism is fairly simple. Rateless Deluge does not require knowledge of the specific packets missed and therefore the transfer of a bit vector of missed packets is unnecessary. Only the *number* of missed packets must be transferred which can be represented as a single byte. This means that for the page size P (in bytes), rateless Deluge reduces the size of the request packet by $\lceil log_2(P-8) \rceil$ bytes for $P > 8$. For $0 \leq P \leq 8$ the packet sizes are the same. This difference results in a slight change to the request suppression method as well. In Deluge, if a node overhears a request packet, it suppresses its own requests if the overheard bit vector is a superset of its own bit vector. Otherwise, the node transmits its own request to the source. In rateless Deluge, a node requests more packets only if it does not overhear another request containing a larger number of requested packets.

The change to the mechanism for transferring data is much more substantial. The original protocol examines the bit vectors it has received and transmits packets corresponding to the union of all those bit vectors. For this, the sending node retrieves a single data packet from Flash memory and transmits it. This process repeats until all the requested packets have been sent. Upon reception of a useful data packet, nodes immediately write that packet to the Flash memory, wait for additional packets, and request retransmission as needed. This model is unrealistic for rateless implementations because the entire page (composed of k individual data packets) is required for the encoding process. Similarly, decoding requires k linearly independent encoded packets to obtain the original page. Therefore significant changes must be made to the page transfer state machine, both at the source and the receiving nodes.

At the source, the transfer state machine must load, encode and broadcast the encoded packets. A simplified state diagram of the new mechanism at the source is given in Figure 3. While it would be possible for the source to load each data packet individually and have the same RAM overhead as the original Deluge, the number of Flash memory accesses would be prohibitive. For a page of k data packets, the source would have to perform k loads for each encoded packet, since each encoded packet is computed as $\sum_{j=1}^{k} \beta_{i,j} X_j$, where X_j represents packet j. Hence, this approach would require $O(k^2)$ Flash accesses to encode each page. A more efficient implementation loads the entire requested page into memory prior to encoding and transmits packets so that only k Flash accesses are required, assuming that a buffer in RAM is large enough to hold the full page. Once the entire page is

[2] Again, more asymptotically efficient row-reduction techniques are known in the literature, but they do not appear practical for implementation in constrained sensor motes.

in memory, the source uses random linear codes to encode and broadcast packets over the wireless channel. With this approach, the number of Flash accesses does not exceed that of original Deluge, and is even potentially lower in the case of retransmissions. Indeed, if a retransmission is required, then original Deluge will have to reload some data packets into the RAM. On the other hand, in the case of rateless Deluge, the working page remains in the buffer. Note, however, that once a different page has been requested, any further request for the previous page will result in additional Flash loads.

Rateless Deluge further changes the transfer mechanism at the source by attempting to anticipate future requests. Once the current request has been fulfilled, rateless Deluge exploits the fact that pages are requested in increasing order by precoding the next page. While other nodes are decoding the previous page, the sending node anticipates requests for the next page, that is, it loads the next page into the page buffer and encode new packets.

However, precoding requires two additional buffers since this process holds multiple encoded packets in memory at once. The process requires a buffer to hold the original page , one to hold the unique identifiers of the encoded packets and another to hold the encoded packets. Encoding on the fly as described above does not require the latter two buffers. This is because once a packet is encoded it is immediately transmitted over the channel and the same memory can be used for the next encoded packet.

Precoding at least partially mitigates the delay associated with encoding at the expense of RAM consumption. That said, in practice, the sending node must wait for a short time period before encoding the next page to avoid excessive Flash accesses which will waste time and energy. For instance, consider the situation in which a single receiving mote needs an additional packet. After a brief time out, this mote will send a retransmission request to the source. If the source begins precoding the next page immediately, the page buffer of the source will contain the next page. Therefore, the request for additional data will require the source to reload the previous page before encoding. This problem can be avoided by introducing a waiting period in between the last data packet transmission associated with the previous page and the start of precoding for the next page.

At the receiving nodes, the mechanism for data reception is changed to allow the nodes to receive k encoded packets and decode the page. A simplified state diagram of the new mechanism at the receiving nodes is given in Figure 4. The rateless version stores each encoded data packet in RAM along with its unique identifier. However, this does not require a new buffer in RAM. Nodes that are receiving packets ignore requests to for data until decoding is complete; this allows the buffer that holds the un-encoded page during encoding to hold the received packets during decoding. Once the number of encoded packets received equals the page size, the unique identifiers are used to re-generate the random matrix and all the packets are decoded. If decoding is successful then the entire page is written to the Flash memory and the page transfer is complete.

However, if there are linearly dependent packets, then in the process of Gaussian elimination there will be rows of all zeros in the β matrix. The packets corresponding to those rows are discarded and additional packets are requested. The same request suppression as described above is used here.

D. ACKless Deluge

Our second rateless OAP protocol is referred to as ACKless Deluge and it attempts to completely eliminate the need for NACKs. ACKless Deluge implements all the structural changes described for rateless Deluge. However, it differs in two significant ways. First, ACKless Deluge employs a FEC mechanism at the packet level which sends extra encoded packets in addition to the requested number of packets. We do not employ bit-level convolutional or block codes to correct bit errors, but instead enhance the rateless features of our implementation by adding redundant packets to account for packet loss. For example, in a trivial system where it is known *a priori* that one packet will be dropped from each page of size k then the FEC mechanism would send $k + 1$ packets. In doing so each receiving node would receive k packets and no retransmission would be required. Our FEC mechanism is designed to prevent requests for retransmission with high probability and how this is accomplished is explored further in Section V. The second major difference is the length of the waiting period associated with precoding. The rationale for a waiting period before precoding is that requests for retransmissions will cause excessive Flash loads. However, since ACKless Deluge prevents (with high probability) retransmissions, the need for a waiting period is largely eliminated. Therefore, ACKless Deluge uses a minimum waiting period before precoding.

E. Overhead

It is clear that rateless Deluge adds memory and computational overhead with regard to original Deluge. The magnitude of this overhead depends on many factors, the most important being the page size. The impact of the computational overhead can be measured in terms of energy consumption, which is explored in Section VII-B, and delay, which is examined here. If there is no packet loss in the network, delay is easily calculated because it depends only on the encoding and decoding times. This is the case because all other factors (i.e. the number of transmissions and the number of Flash reads and writes) are identical. Figure 5 depicts the delay incurred with four different schemes: i) without precoding, ii) with precoding, but with no waiting period, iii) with precoding and 0.5 second waiting period, and (iv) with precoding and a 2 seconds waiting period. In each case, the figure shows the amount of time needed to encode and decode 48 packets for varying page sizes. The length of the waiting period determines whether or not precoding is beneficial or not. When there is a minimum waiting period (as is the case with ACKless Deluge), precoding completely eliminates the time overhead due to encoding. However, with increasing waiting periods and

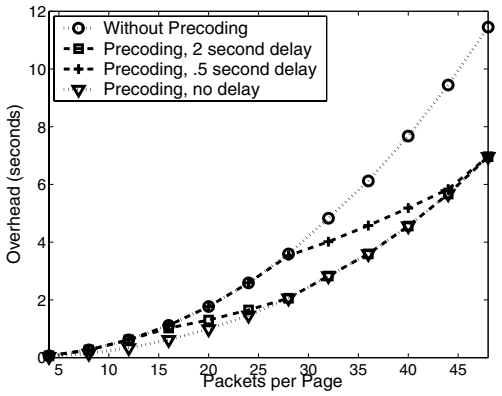

Fig. 5. Overhead (encoding and decoding) of rateless Deluge, with and without precoding and with and without waiting period. The figure shows that precoding can reduce the time overhead of rateless Deluge, but the precise amount depends on the waiting period and the page size.

page sizes, some packets (possibly all) will not be encoded during the precoding stage. For example, Figure 5 shows that with a 2 seconds waiting period, no benefit is achieved from precoding if the page size is less than 28 packets, because the decoding process on the receiving nodes is shorter than the waiting period.

The RAM memory overhead shows a similar trend: the amount of extra memory consumed increases with the page size. The major sources of increased RAM usage are the table of multiplicative inverses, the page buffer, the precoding buffer, and the buffer of unique identifiers. While the size of the table of inverses is constant at 256 bytes, the size of each of the other buffers is a linear function of the page size. For a page size of 20 packets per page this overhead translates into 1196 bytes in RAM which represents 11.6% of the 10 KB of RAM on a Tmote Sky mote. When the page size increases to 48 packets per page the additional cost is 24.5% of total RAM.

The costs of rateless Deluge which are outlined above imply that a page size that is smaller than the default Deluge page size of 48 packets per page would be beneficial. However, there is a trade off because at some point the reduction in the page size becomes counter productive. To illustrate this consider the extreme case where the page size is 1 packet; both rateless and original Deluge will transmit a single packet upon a new request and the benefit of rateless Deluge is lost. While small page sizes greater than one will show some benefit, higher page sizes increase the communication and energy gains of rateless coding, as shown by our experiments and simulations in the sequel. Figure 5 provides another justification for using a page large enough. This figure shows that when there is a waiting period before precoding, then a reduction in the overhead is only seen when the pages size exceeds a certain threshold. For these reasons, all experiments are performed with a page size of 20 packets.

V. Numerical FEC for ACKless Deluge

The FEC mechanism in ACKless Deluge operates at the packet level. It sends extra encoded packets to prevent the need for additional control messages and retransmission with high probability. In [10], it is suggested that extreme value theory can be applied to determine the number of extra transmissions required. Using knowledge of the number of recipients (N), the number of packets required (M), and an estimate of the loss probability (p) it is shown in Theorem 11 that the number of transmissions is bounded by random variables converging to the cdf of a normalized Gumbel distribution as $N \to \infty$. A Gumbel distribution is of the form $G(x)= \exp(-\exp(-x)$. Simulation in [10] shows that even though the convergence is known only as $N \to \infty$, the number of transmissions is close for relatively low N; the simulation uses the example of $N = 100$. However, the formula of Theorem 11 provided in [10] may not be accurate enough in sparse networks where N can be very small.

Clearly, the FEC mechanism in ACKless Deluge should be flexible so that all values of N can be accommodated. To allow for an accurate answer for all N, ACKless Deluge numerically computes the solution for small values of N, as explained next. In [10], the time for user n to receive M packets, denoted by the variable T_n, is shown to be negative binomially distributed. By definition, $P(T_n \leq t) = F(t) = I(1-p, M, t-M)$ where $I(x, a, b)$ is the regularized beta function. If you assume that each node loses packets independently of all other nodes, the probability that the maximum is less than t is the following:

$$\Pr(\max_{i=1\ldots N} T_i \leq t) \quad = \quad (F(t))^N$$

By selecting t appropriately high, one can determine the number of extra transmissions needed to guarantee success with high probability.

Similarly to extreme value FEC, knowledge of N and p is necessary for the computation. A conservative estimate of the packet loss can be obtained by active or passive probing. For instance, a passive probing approach is to periodically transmit pages without extra packets. The number of packets requested for retransmission can be then used to estimate the packet loss probability. The number of neighbors can easily be determined by keeping a table of known neighbors based on overhead advertisements and requests. Advertisements by nodes happen systematically regardless of the amount of data present on the nodes and are therefore a good way to gain an estimate of the number of nodes in range. To accommodate network topology changes, the table could be refreshed periodically. The accuracy of the estimates of N and p has an effect on the amount of communication required. An overly conservative estimate will cause excessive transmission of data packets and a low estimate will cause additional requests. The effects of using numerical FEC in sparse networks and accurate estimates of the parameters N and p are explored in Section VI-C.

VI. Experimental results

The experiments in this section evaluate the performance of rateless Deluge and ACKless Deluge with respect to the

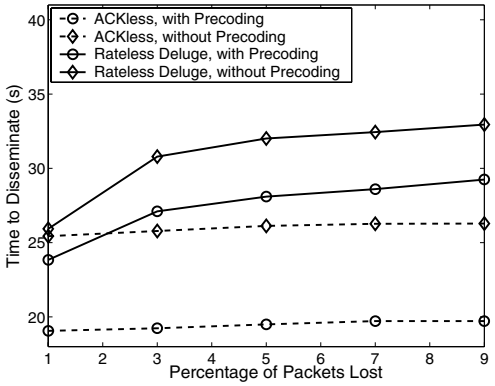

Fig. 6. Average dissemination time of rateless Deluge and ACKless Deluge, as a function of the packet loss. Precoding reduces the dissemination time over a single hop for both rateless versions.

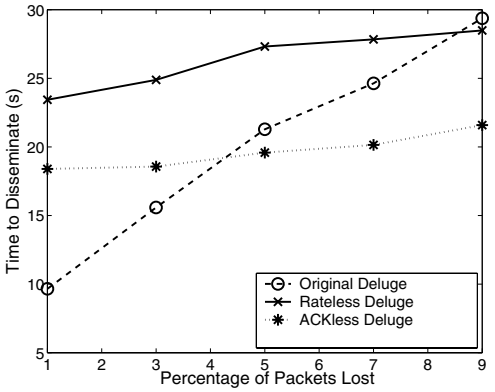

Fig. 7. Comparison of average dissemination time of a 9-pages image with increasing packet loss for each protocol.

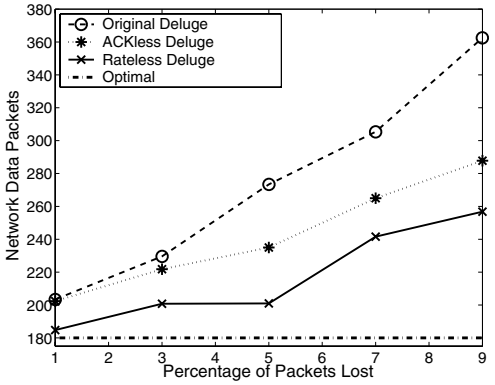

Fig. 8. Average number of packets transmitted on the data plane as a function of the packet loss for each protocol.

original Deluge protocol. All the experiments are performed on a test bed of 2.4 GHz Tmote Sky motes. Packet loss is generated in two different ways depending on the experiment: i) forced packet loss, which is the practice of dropping packets uniformly at random at a certain rate; in this case the motes transmit at their highest power setting over short distances to ensure a good link; and ii) natural packet loss, which induces packet loss by transmitting at the lowest power setting in the presence of interference. Interference is provided by motes transmitting over the advertisement, data, and control channels; the delay between transmissions is chosen uniformly at random between 3 and 7 milliseconds.

A. Single Hop

All single-hop experiments are performed on a 16-motes testbed that form a fully connected graph. At the beginning of the experiment a 9-pages image is present on one mote (i.e., the base station). The experiment ends once every mote has the entire image in Flash. Data is collected in two ways. A single mote is connected to a PC and records all network activity and each node in the network logs local statistics and stores them to Flash memory. Each data point represents the average over five trials. The experiments are conducted under forced packet loss.

Two single hop experiments are performed. The first experiment shows the benefits of precoding, namely, the practice of anticipating future requests to mitigate the cost of encoding. Figure 6 shows that for both rateless and ACKless Deluge precoding significantly decreases the time to disseminate a file over one hop; for example, Figure 6 shows that when 7 percent of packets are lost, precoding reduces the time to disseminate by 9.1 percent for rateless Deluge and 24.9 percent for ACKless Deluge. The reduction for ACKless Deluge is much larger because it assumes no additional transmission is needed and it does not wait for requests before starting to precode. Looking at the specific numbers at 7 percent loss, ACKless Deluge is 6.5484 seconds faster with precoding than ACKless Deluge without precoding. At 20 packets per page, the cost to encode a single packet is on average .03897 seconds and the encoding cost is 7.0146 for the entire 9-pages file. In this case, ACKless Deluge has mitigated 93.4 percent of the encoding overhead. This confirms the calculation of Section IV-E that indicated that precoding mitigates the overhead of encoding. In fact, for the case of ACKless Deluge, the overhead almost completely disappears.

The second experiment compares the original, rateless, and ACKless Deluge protocols in terms of the time to disseminate the image, the number of data packets transmitted, and the number of requests transmitted in the network. Figure 7 shows the amount of time in seconds to disseminate the entire image to all nodes. Since this experiment is performed on our testbed, these times include all computational overhead of our implementation. With minimal packet loss, original Deluge performs significantly better than either version of rateless Deluge. However, as the packet loss increases, original Deluge becomes worse than ACKless Deluge starting from a packet loss rate of 4.5 percent and than rateless Deluge starting from a packet loss rate of 8.5 percent. Rateless Deluge always performs worse than ACKless Deluge because of the waiting period it uses to prevent excessive Flash loading.

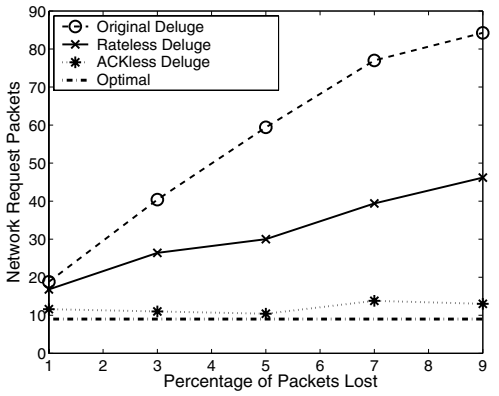

Fig. 9. Average number of packets transmitted on the control plane as a function of the packet loss for each protocol.

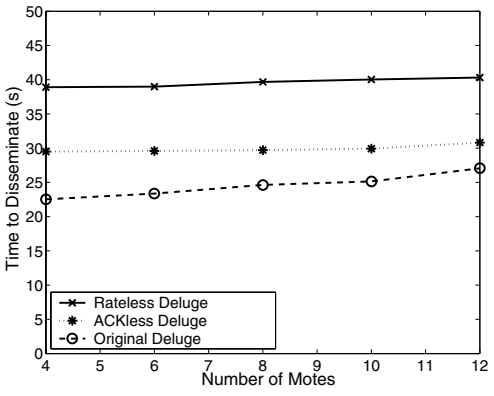

Fig. 10. Average time to disseminate an image over a two-hop network for each protocol, as a function of the network size.

The transmissions in the data plane for each version as well as the minimum number of transmissions are shown in Figure 8. The minimal number of packets is 180 since there are 20 data packets per page and 9 pages in the image. At low packet loss, rateless Deluge performs near optimal while the other two versions perform similarly. However, as packet loss increases, original Deluge requires significantly more data packets to disseminate the image. As expected, rateless Deluge transmits less data packets than ACKless Deluge.

The transmissions in the control plane for each version and the minimal number are shown in Figure 9. The minimum number of packets is 9, one for each page in the image. In this case, original Deluge always performs worse than both rateless versions. ACKless Deluge uses near optimal transmissions for all rates of packet loss and rateless Deluge performs in between ACKless and original Deluge.

The conclusions to be drawn from the single hop experiments are that with increasing packet loss rateless and ACKless Deluge both perform better than the original version in communication complexity. Furthermore, as packet loss increases the rateless versions disseminate the image in less time; the reductions in the communication on the data and control planes of the rateless version are significant enough to overcome the inherent overhead of rateless codes. When comparing the two rateless protocols, ACKless Deluge shows significant reduction of transmissions in the control plane and rateless Deluge performs better in the data plane. This is an expected result because ACKless Deluge adds extra data packets to eliminate retransmission while rateless Deluge only sends the minimum amount of data packets requested.

B. Multi-Hop

The multi-hop experiments are performed with a varying numbers of motes and natural packet loss. In each experiment, one mote possesses the entire 9-pages image, $\frac{N}{2}$ motes are placed a single hop away from the source, and $\frac{N}{2}$ are placed two hops away from the source. Each data point represents an average over five trials. Once again, the data is collected in two ways. A single mote is connected to a PC and records all network activity and each node in the network logs local statistics and stores them to Flash memory.

The multi-hop experiment compares original, rateless, and ACKless Deluge in terms of the time to disseminate the image, the number of data packets transmitted, and the number of requests transmitted in a *low density* network. Figure 10 shows the amount of time in seconds to disseminate the image over two hops; in the figure, the number of motes shown on the x-axis is divided evenly between the two hops. The figure shows that both rateless Deluge and ACKless Deluge perform slower than original Deluge over two hops. However, as the number of motes increases the average dissemination time of both rateless versions increases noticeably slower than that of original Deluge. This trend suggests that at higher density the rateless versions would disseminate the image faster, unfortunately our testbed is not large enough to verify this conjecture. Scalability to higher density networks is explored further in Section VII-A.

The results for the communication complexity on data and control planes show that, even at low density, the rateless versions significantly reduce the amount of transmissions on the multi-hop network. Both rateless protocols transmits fewer data packets than original Deluge as the number of receiving motes increases. Similarly to single hop, rateless Deluge transmits fewer data packets than ACKless Deluge. Both rateless versions perform far better than original Deluge in the control plane, and once again, ACKless Deluge is near optimal.

C. FEC

As explained in Section V, using numerical FEC is appropriate since the extreme value theory bound of [10] is not accurate for small N. For these experiments we have selected $\Pr(\max_{i=1\ldots N} T_i \leq t) = .95$. The number of extra packets sent with each transmission is dependent on the number of neighbors and the probability of loss. Figure 11 shows the reduction in data packets transmitted to distribute a 9-pages image in sparse networks. The experiments uses forced

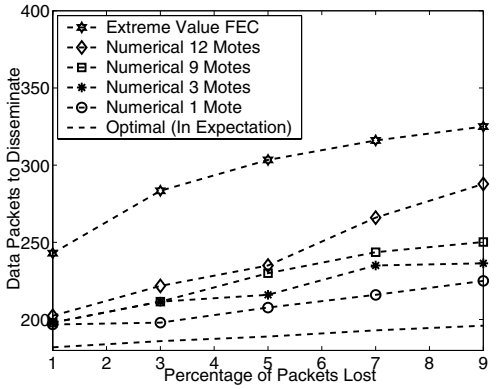

Fig. 11. Comparison of numerical FEC and extreme value FEC [10] in terms of data packet transmissions versus packet loss. Numerical FEC allows for reduced communication of the data plane.

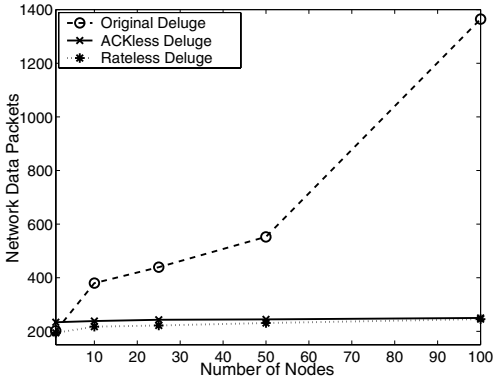

Fig. 12. Average number of packets transmitted on the data plane as function of the network density, for each protocol.

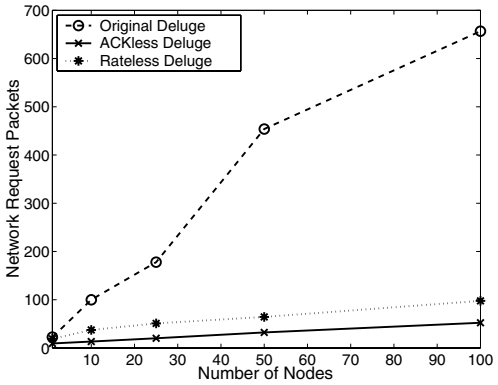

Fig. 13. Average number of packets transmitted on the control plane as a function of the network density, for each protocol.

packet loss and varying network densities and shows that the numerical method reduces the amount of data transmitted significantly compared to the analytical one. Additionally, as would be expected, increasing network density causes the number of packets to be sent to converge towards the analytical expression employed by the extreme value approach.

Section V also indicates that ACKless Deluge requires knowledge of two parameters: the probability of loss, p, and the number of neighbors, N. To explore the sensitivity to these parameters consider a network in which the source has 8 single hop neighbors and the network has a loss probability of 5%. In this case if p is assumed to be correct and N is slightly overestimated as 12 the number of data packets increases by 4.2% and the number of request packets is the same. However, if N is estimated as 100 then the number of data packets increases by 32%. When N is underestimated as 1, the number of data packets decreases by 2.4% and the number of requests increases by 38%. This shows that significantly over estimating N causes a large increase in the number of data packets, but only a small under estimation causes a significant increase in the number of control packets sent. If N is assumed to be correct and p is overestimated as 9% packet loss, the number of data packets increases by 14.3% and the number of requests is approximately the same. When p is underestimated as 1% packet loss the number of data packets decreases by 12.7% and the number of requests increases by 7.7%. These examples reflect large error in the estimate of N and p, but it should be noted that the algorithm is tolerant to small errors in the estimate. Overall, when the goal of using the protocol is to reduce the amount of transmission in the control plan, a overestimation of either parameter increases the number of data packets sent, but maintains the amount of control packets sent.

VII. Simulation

A. Scalability

The preceding experiments have shown that at low network densities and packet loss, rateless Deluge transmits fewer data and control packets than original Deluge. However, to be an effective solution, rateless Deluge must also scale well with increasing network density. To simulate these conditions we have used the TinyOS simulator, TOSSIM, and configured it so that all motes are within one hop of the source and packets are dropped at a rate of 7 percent. Only the numbers of data and control packets transmitted have been collected for these simulations. No timing data has been collected because TOSSIM considers all processing to happen instantaneously. This does not lend itself to a fair comparison since all of the overhead of rateless Deluge is due to processing and, thus, would be ignored.

Figure 12 shows the number of data packets transmitted to disseminate a 9-pages image at varying network densities. At low density the simulation performs similarly to the experiments on the motes; rateless Deluge has near optimal performance while ACKless and original Deluge perform similarly. However, as the number of receiving nodes increases, original Deluge rapidly increases the amount of packets it sends while ACKless Deluge keeps transmitting a consistent amount. The amount transmitted by rateless Deluge appears to converges to that by ACKless Deluge. The number of control plane packets

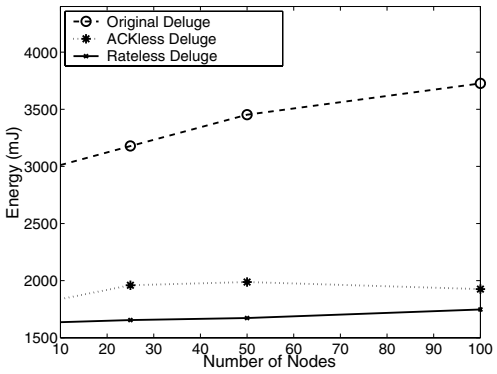

Fig. 14. Average energy use per node over a single hop with increasing network density.

transmitted is shown in Figure 13. As the density increases, the amount of control packets transmitted by the original version of Deluge, increases rapidly, while the amount transmitted by rateless and ACKless Deluge increases much more slowly.

B. Energy Savings

The amount of energy used to disseminate an image is another very important metric that can be used to compare the behavior of the different protocols. Indeed, wireless sensor networks are (generally) powered by batteries and lower energy usage will extend the lifetime of the network. To get an idea of the energy savings over a single hop, simulations are run using the power modeling capabilities of PowerTOSSIM, an extension of TOSSIM. Since PowerTOSSIM does compute processing energy it provides a valid comparison. In our simulation, a 9-pages image is disseminated to a varying number of modes over a single hop with a packet loss rate of 7 percent.

The results of our simulation are presented in Figure 14, which shows the average energy consumed in millijoule (mJ) per node at different network densities. The total energy of the original protocol is substantially larger than each of the rateless versions. This is because the original protocol expands a larger amount of communication on both the data and control planes. At lower densities, rateless Deluge performs better than ACKless Deluge, but as the number of nodes increases the energy use of each rateless protocol begins to converge.

VIII. Conclusion

In this paper, we have shown the benefits of using random linear codes for over-the-air programming of sensor networks. Compared to Deluge, one of the most widely used OAP protocol at present, our implementations (i) reduce communication on both the data and control planes, (ii) reduce latency at moderate levels of packet loss, (iii) are more scalable to dense networks, and (iv) generally consume far less energy, a premium resource in wireless sensor networks. We have presented two rateless OAP protocols, namely rateless Deluge and ACKless Deluge. Although ACKless Deluge adds communication on the data plane, it is particularly efficient on the control plane as it almost completely eliminates the needs for retransmission requests by receiving nodes and packet retransmissions by sources. Since it unlikely that nodes will request packets belonging to a previous page, ACKless Deluge is able to take full advantage of precoding and speed-up data transfer. We have provided a simple mathematical approach to determining the number of extra packets needed by ACKless Deluge in order to guarantee, with high probability, that all the nodes receive enough packets to decode a page. Overall, this work has shown that rateless Deluge, augmented with our FEC mechanism, achieves excellent performance with respect to almost all the metrics relevant to wireless sensor networks. More generally, we expect rateless code transfer mechanisms, similar to these presented in this paper, to be practical and useful for any communication protocol in wireless sensor network that must overcome traffic congestion and packet loss due to packet broadcast.

IX. Acknowledgements

The authors wish to thank the reviewers for their editorial inputs. This work is partially funded by the National Science Foundation under grants CNS-0435312 and CCF-0729158 and by a grant from Deutsche Telekom Corp.

References

[1] A. Chlipala, J. Hui, and G. Tolle. Deluge: Data dissemination for network reprogramming at scale. Class Project, http://www.cs.berkeley.edu/~jwhui/research/deluge/cs262/cs262a-report.pdf, Fall 2003.

[2] J. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys'04*, Baltimore, Maryland, USA, Nov. 2004.

[3] S. Kulkarni and M. Arumugam. Infuse: A TDMA-based data dissemination protocol for sensor networks, November 2004. Technical Report MSU-CSE-04-46, Department of Computer Science, Michigan State University.

[4] S. Kulkarni and L. Wang. Mnp: Multihop network reprogramming service for sensor networks. In *25th IEEE International Conference on Distributed Computing Systems*, pages 7–16, 2005.

[5] P. Levis and D. Culler. Mate: A virtual machine for tiny networked sensors. pages 85–95, Oct. 2002.

[6] T. Liu and M. Martonosi. Impala: A middleware system for managing autonomic, parallel sensor systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, June 2003.

[7] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for wireless embedded devices. In *26th IEEE Real-Time Systems Symposium*, 2005.

[8] N. Reijers and K. Langendoen. Efficient code distribution in wireless sensor networks. In *WSNA*, 2003.

[9] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[10] D. Starobinski, W. Xiao, X. Qin, and A. Trachtenberg. Near-optimal data dissemination policies for multi-channel, single radio wireless sensor networks. In *IEEE INFOCOM*, 2007.

[11] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, 2003.

[12] Q. Wang, Y. Zhu, and L. Cheng. Reprogramming wireless sensor networks: Challenges and approaches. *IEEE Network Magazine*, 20(3):48–55, May-June 2006.