Reliable Transfer on Wireless Sensor Networks

Sukun Kim
Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720
Email: binetude@eecs.berkeley.edu

Rodrigo Fonseca
Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720
Email: rfonseca@eecs.berkeley.edu

David Culler
Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720
Email: culler@eecs.berkeley.edu

***Abstract*— Many applications in Wireless Sensor Networks, including structure monitoring, require collecting all data without loss from nodes. End-to-end retransmission, which is used in the Internet for reliable transport, becomes very inefficient in Wireless Sensor Networks, since wireless communication, and constrained resources pose new challenges. We look at factors affecting reliability, and search for efficient combinations of the possible options. Information redundancy like retransmission, and erasure codes, can be used. Route fix, which tries alternative next hop after some failures, also reduces packet loss. We implemented and evaluated these options on a real testbed of Berkeley Mica2Dot motes. Our experimental results show that each option overcomes different kinds of failures. Link-level retransmission is efficient but limited in achieving reliability. Erasure code enables very high reliability by tolerating packet losses. Route fix responds to link failures quickly. Previous work had found it difficult to increase reliability past a certain threshold. We show that the right combination of primitives can yield more than 99% reliability with low overhead, providing a viable alternative to end-to-end retransmission over multiple hops.**

I. Introduction

There exist many applications in Wireless Sensor Networks requiring all data to be transmitted without loss. For example, structure monitoring needs the entire data from all measuring points to build a model and analyze it. Moreover, data collection can be done over multi-hop network.

Challenges to achieving reliability on Wireless Sensor Networks can be divided to three main categories. First problems are related to the wireless communication [1], [2]. The asymmetry of links makes link quality estimation hard and invalidate many assumptions made in other environments. Correlated losses due to obstacles, interference, can lead to consecutive losses, decreasing the effectiveness of erasure code. Weak correlation between quality and distance, hidden terminal problems, and dynamic change of connectivity complicates the situation further.

The second sort of problems comes from the constrained resources of Wireless Sensor Networks motes. A mote is battery powered, so has a limited power source. It also has small computational power and memory space. Furthermore, its communication bandwidth is narrow. Therefore we can't run a complicated algorithm to achieve reliability: the algorithms run on motes should not send too much overhead traffic, and should not be computationally or storage intensive.

Finally, from a software engineering standpoint, diverse routing layers add more challenges. Since motes are resource constrained, applications tend to make heavy use of customization and cross-layer optimizations [3]. Therefore, there are different routing layers customized for specific purpose: even if we can use a general purpose, point-to-point routing for dissemination of information or collection of data, this approach is very inefficient for some specific cases. For collecting data (convergence routing), each node only needs to keep track of which nodes are candidates for its parent. This reduces the burden of keeping additional information to support routing to any node. Dissemination of information, such as code image distribution, is similar to multicast (divergence routing). In this case, we can benefit from the broadcast nature of wireless communication. By injecting one packet into channel, all neighboring nodes can hear the packet. Compared to sending packet to each single receiver, this can save a huge effort. So there are three main routing layers categories: *point-to-point routing*, *convergence routing*, and *divergence routing*. One transport layer or one method may not work for all three cases well. But it is not a good idea to keep three separate versions of reliable transfer either. At least it will be desirable to share some components if possible, wherever it might be located in network stack. Ultimately, we seek to find common reliability primitives or principles that can be used even in different routing layers.

In this paper, we examine diverse options for improving reliability over multiple-hops, focusing mainly on point-to-point routing. First of all, it is worthwhile looking at fundamental factors that determe reliability. Then we look at possible options which improve each factor. Let us simplistically look at the following equation

$$\text{number of packets received} = P_{success} \times \text{number of packets sent}$$

The goal is to increase 'number of packets received' sufficiently so that we can get all data. Even though it is also important which packets are received, as we will see later, the basic limitation is delivering a sufficient amount of packets. This in turn amounts to increasing either 'number of packets sent' or increasing the probability to get through '$P_{success}$'.

Increasing the number of packets sent can be interpreted as adding redundancy to information. One option is retransmission. End-to-end retransmission is used in TCP on the Internet

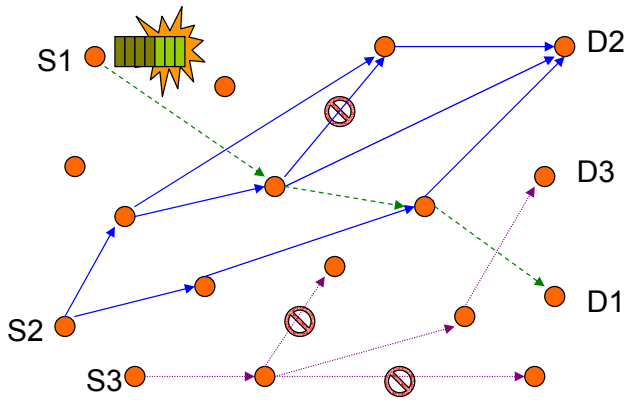

Fig. 1. Possible options to achieve reliability. S1 uses erasure code producing additional code words, S2 uses thick (multiple) path which is not examined in this work, and S3 do route fix finding alternative next hop when stuck

[4]. Link-level retransmission is used in wireless communication where loss rate of link is high. Adding redundant data is also an option. Sending an additional parity packet for some number of previous packets is a good example. Erasure codes can be thought as a generalization of parity code. Rather than sending one additional packet, erasure code can send multiple additional packets. In parity packet case, any M out of $M+1$ packets will reconstruct original M data. Likewise, erasure code enables reconstruction of M original data packets if any M out of $M+R$ packets are received. In Figure 1, $S1$ is sending data with erasure code. We can also exploit spatial redundancy along the path. As $S2$ in Figure 1, 'thick path' can be used as in [5]. Every node within nearby area along the path will participate in transferring data. This method adds in-network data redundancy.

Increasing the probability of successful delivery and changing the loss distribution can solve problems which are hard to overcome by redundancy alone. Let us assume $P_{success}$ is not randomly distributed. Erasure code can survive up to R losses. When consecutive $R+1$ or more packets are lost, erasure code is unable to reconstruct the original data. This phenomenon happens in wireless communication. For example, after a link failure, it takes time for the routing table to be updated. Until then, all packets sent to that link will fail, introducing consecutive failures. In this situation, we can quickly try an alternative next hop. This is shown as $S3$ in Figure 1.

In this paper, we look at link-level retransmission, erasure code, and alternative route. Other possible options like thick path and end-to-end retransmission remain as future work. We examine several options on real-world testbed. We provide results in Section VII. We then see which options and which combinations thereof are good choices.

II. Related Work

There are many algorithms proposed and implemented for multi-hop communications in sensor networks (e.g. [2], [6]–[12]) , and as noted these can be broadly divided in convergence, divergence, and point-to-point. Our work is orthogonal to these routing implementations, as we examine techniques that can be employed to varying degrees in most multihop routing schemes. In particular, we show that it is a good feature of a routing algorithm to provide alternative next hops towards a destination.

Previous work has been done in reliable transport for sensor networks. PSFQ [13] examines the problem of retasking a sensor network (an example of divergence) reliably, and make use of hop by hop recovery with caching at intermediate nodes, as opposed to end-to-end recovery. RMST [14] investigates through simulation the tradeoff between having reliability implemented at the MAC, transport, and application layers. Both works conclude that hop by hop recovery is very important for achieving reliability and that end-to-end recovery is not adequate. They only consider different retransmission/repair options and use simulated data. Our contribution to their findings is the addition of the very effective options of erasure coding and alternate route for providing reliability, as well as the examining the interaction of these different mechanisms. We also use real implementation of the options on a testbed of wireless motes, which allows us to see the effect of the radio environment on the reliability.

There exist diverse algorithms for erasure coding which can be implemented in either software or hardware [15], [16]. [15] exploits diverse optimizations, from which this work gained many hints. It is an excellent introduction to Reed-Solomon codes, but focuses on the implementation in desktop computers. We leverage many of its optimizations, carefully choosing parameters suitable for very resource constrained WSNs. Rateless codes [17], [18] is a class of erasure code in which arbitrary number of code words can be produced, and is optimized for delivery of very large amounts of data over high bandwidth, high latency Internet links. These works are not optimized for systems with low capability: not much attention was paid to cases with extreme space limitation. Work in this paper puts heavy weight on optimization for nodes with very limited resources.

III. Link-level Retransmission

The loss rate on wireless links is much higher than that of wired links, and this effect accumulates quickly as the number of hops increases. For example, when loss rate is 10% per hop, after 15 hops loss rate becomes 80%! If a message is lost at the n^{th} hop, all previous $n-1$ transfers become wasted effort. To deliver the packet to n^{th} hop again, we need $n-1$ additional transfers, if all $n-1$ transfers succeed. With link-level retransmission, just one retransmission can bring packet to the same point. For efficient use of the wireless channel, link-level retransmission is a very attractive choice.
There are drawbacks in link-level retransmission, when used in some specific contexts. When retransmission is implemented with link-level acknowledgments, there is a decrease in channel utilization. This has been measured to be as high as 20%

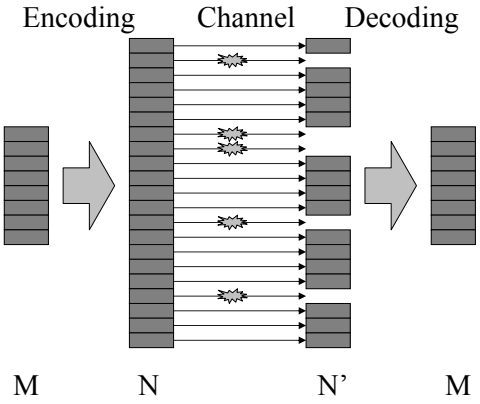

Fig. 2. Mechanism of Erasure Code

in TinyOS.[1] This overhead can, however, be mitigated in some contexts by using techniques such as passive acknowlegments, in which the next hop transmission is interpreted as an acknowledgment. Another minor downside is that the middle node needs to hold the packets in a buffer until it receives acknowledgement from the next hop. Lastly, the delivery time depends on the number of retransmissions along the route, so the ent-to-end round trip time (RTT) can vary significantly. This situation makes end-to-end retransmission inefficient. Since we do not clearly know the RTT, an (over estimated) upper bound needs to be used. The sender holds its buffer for a longer time than necessary. Holding memory space for a long time is not desirable in resource-constrained Wireless Sensor Networks.

IV. Erasure Code

Another important mechanism we employ is erasure coding. It is a scheme with which we can reconstruct m original messages by receiving any m out of n code words ($n > m$). If n is sufficiently large compared to the loss rate, we can achieve high reliability without retransmission. Figure 2 shows high level mechanism of erasure code. We use a particular erasure coding algorithm, Reed-Solomon coding. Before we explain the Reed-Solomon code, we first introduce linear codes and Vandermonde matrices.

A. Linear Code

For the encoding process, an encoding function $C(X)$ is used, where X is a vector of m messages. $C(X)$ produces a vector of n code words ($n > m$). If the code has the property that $C(X) + C(Y) = C(X + Y)$, then it is called a linear code. Linear codes can be represented by a matrix A, and encoding can be represented by a matrix-vector multiplication: the code word vector Z for message vector X is simply AX. Decoding entails finding X such that $AX = Z$, for a received code word vector Z, *i.e.* finding the solution to the linear equation $AX = Z$. We can see that A should have m linearly independent rows so that the linear equation has a unique solution, which represents a unique message vector.
This code is very useful in practice, since encoding and decoding are computationally inexpensive, and this is especially so in resource-constrained Wireless Sensor Networks.

[1]The MAC layer in TinyOS waits 20% of a packet time before considering the transmission successful.

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{m-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{m-1} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{m-1} \\ 1 & x_n & x_n^2 & \cdots & x_n^{m-1} \end{pmatrix}$$

Fig. 3. Vandermonde Matrix

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{m-1} \\ 1 & x_2 & \cdots & x_2^{m-1} \\ 1 & x_3 & \cdots & x_3^{m-1} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{m-1} \\ 1 & x_n & \cdots & x_n^{m-1} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{m-1} \end{pmatrix} = \begin{pmatrix} p(x_1) \\ p(x_2) \\ p(x_3) \\ \vdots \\ \vdots \\ p(x_{n-1}) \\ p(x_n) \end{pmatrix}$$

Fig. 4. High level diagram showing how Reed-Solomon code works

B. Vandermonde Matrix

There is one more thing we need to look at before describing Reed-Solomon codes. A Vandermonde matrix is a matrix with elements $A(i, j) = x_i^{j-1}$, where each x_i is nonzero and distinct from each other, as shown in Figure 3. For an n *by* m Vandermonde matrix ($n > m$), any set of m rows forms a non-singular matrix. For whatever set with m rows we may choose, rows in the set are linearly independent. Let's define for future reference this property as $Property\ V$.

Definition ($Property\ V$): For an n *by* m ($n > m$) matrix A, if any set S of m rows of A form non-singular matrix such that all rows in S are linearly independent, then A is said to have $Property\ V$.

Vandermonde matrices have $Property\ V$. So we can see that in a linear equation $AX = Z$, where A is a Vandermonde matrix, any m rows and corresponding m elements of Z form an m *by* m square matrix and a vector of size m, where the matrix is non-singular. Then we can uniquely determine X. This is a key property used in our implementation of the Reed-Solomon code.

C. Reed-Solomon Code

The basic idea of Reed-Solomon code is to produce n equations with m unknown variables ($n > m$) such that with any m out of n equations, we can find those m unknowns.
For some given data, let us break it down into m messages $w_0, w_1, w_2, \ldots, w_{m-1}$, and construct the polynomial $P(X)$

using these messages as coefficients, such that

$$P(X) = \sum_{i=0}^{m-1} w_i x^i$$

We then evaluate this polynomial $P(X)$ at n different points $x_1, x_2, , x_n$. $P(x_1), P(x_2), ..., P(x_n)$ can be represented as multiplication of a matrix and a vector, as shown in Figure 4.

Here we can see that matrix A is a Vandermonde matrix, W is a vector of messages, and code words are contained in a vector AW. If we have any m rows of A and their corresponding $P(X)$ values, we can obtain the vector W which contains coefficients of the polynomial, which is again the original messages. Reed-Solomon codes can also be used to correct errors. However, in current implementation of TinyOS, each packet has a CRC to detect bit errors. We can assume that there will be no bit errors in packet containing code words, as these are dopped by the lower layers. Therefore, error correction is not used in the implementation.

V. Modifications for Wireless Sensor Networks

There are modifications needed to bring erasure codes to a real world implementation, especially in resource-constrained Wireless Sensor Networks (WSN). Several methods used to improve efficiency in motes are heavily borrowed from [15]. We need an efficient representation of the data and efficient and precise operations, including vector arithmetic and matrix inversions. Fortunately, these can be made very efficient with modular operations on finite fields and clever lookup tables, which we discuss next.

A. Extension Fields

To make efficient use of bits in the packet, maintain precision, and reduce computational effort, we do all calculations in an *extension field* with base 2. We briefly introduce fields, and prime fields. A field [19] is any set of elements with two operations addition and multiplication, that satisfies the *field axioms* – commutativity, associativity, distributivity, identity, and inverses – for both operations. Every nonzero element has an inverse.

A field with a finite number of members is known as a finite field or Galois field. For a given Galois field of size q, if $q-1$ powers of an element x $(x^1, x^2, \ldots, x^{q-1})$ produce all non-zero elements, that element x is called a generator of the given Galois field.

A *prime field* is a Galois field whose elements are integers in $[0, p-1]$, where p is prime. Addition and multiplication are normal integer addition and multiplication with modulo operation at the end. Prime field always have a generator. The size of prime field is p, and we need $\lceil \log_2(p) \rceil$ bits to represent all elements. Since p is not power of 2, there's waste in bit usage. For example, to represent prime field with prime 11, we need 4 bits with which 16 numbers can be represented.

An *extension field* is a Galois field whose elements are integers in $[0, p^r - 1]$. Extension fields can be though as polynomials on prime field(p). Operations follow the rules of polynomial operation with modulo operation at the end. A primitive polynomial is the generator of extension field. Interestingly, this set with polynomial operations stated above still satisfies the properties of fields. Moreover, by setting $p = 2$, we can fully utilize bits in message. $Property\ V$ of Vandermonde matrices still holds for prime field, and even for extension fields!

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & x_{m+1} & \cdots & x_{m+1}^{m-1} \\ 1 & x_{m+2} & \cdots & x_{m+2}^{m-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^{m-1} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{m-1} \end{pmatrix} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{m-1} \\ p(x_{m+1}) \\ p(x_{m+2}) \\ \vdots \\ p(x_n) \end{pmatrix}$$

Fig. 5. Systematic code

B. Systematic Code

When coding a message, if part of the encoded message is the original message itself, it is possible to recover the original message without decoding, in the event that this part arrives intact. Codes with this property are called systematic codes. Another good property of Vandermonde matrices is that if any m rows of an n *by* m $(n > m)$ Vandermonde matrix are substituted with rows of the m *by* m identity matrix, the new matrix still has $Property\ V$, even for extension fields. Figure 5 shows one possible case. This matrix will clearly produce a systematic code, as m of the code words will be the original message. When we use a systematic code in this way, at the encoding side, we don't need any computation for the portion of code words containing original messages. Systematic codes can give a benefit even when we loose some packets. At the decoding side, the more of the original message part we have, the closer the decoding matrix is to the identity matrix, and the quicker the decoding process becomes.

C. Multiple Independent Code Words in a Packet

If one packet carries one code word, each code word will be very large. This makes the implementation intractable since operations on such a large field require huge space and time. One solution would be to use small messages and small code words. Then, however, the payload in a packet gets too small. By putting multiple independent code words into a packet, we can fully utilize payload space of a packet without problems of large code word.

Imagine dividing one big data into t small pieces of data. Then each data is again divided into m messages, and encoded into n code words. We have total of tn code words to send. Pack the i^{th} code words from each independent k data into a single packet. We either get all i^{th} code words for k data, or we get nothing. Any m packets will provide m code words for all k data, and we can reconstruct original k data. Since all

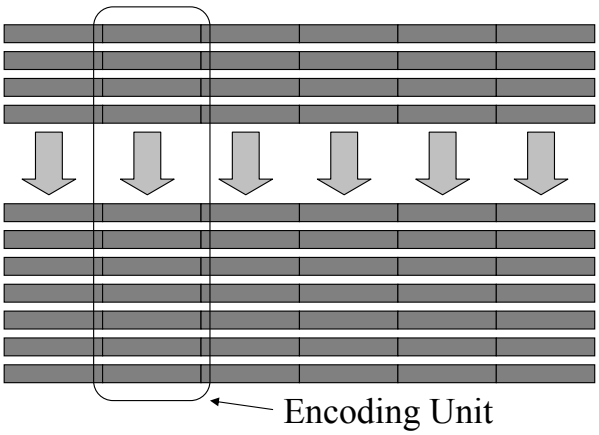

Fig. 6. Divide packet into multiple independent code words

k data have code words with same sequence set, decoding process is the same: the same decoding matrix can be used. This further enhances decoding efficiency by amortizing the matrix inversion cost over k data packets. Figure 6 shows example. Here data is divided into 6 small data chunks. Each data chunk is divided again to 4 messages. Messages from each chunk are encoded to 7 code words independently. Then code words from all data chunks with same sequence number are packed into the same packet.

The drawback of dividing packets into multiple code words is the contraints on the number of messages and code words. The number of messages can not exceed the number of bits used to represent the message. The number of code words should be smaller than the size of extension field. For example, if each code word is 8 bit long, maximum number of messages is limited to 8, and maximum number of code words is limited to 255.

D. Operation Table

Operations on extension fields are not simply addition and multiplication combined with modulo operation. They are polynomial operations with modulo. Therefore, rather than performing complex computation on the fly, we use lookup tables. Addition is simply the XOR of two numbers, and we don't need a table. For multiplication and division, exponent and log values are computed and stored as tables.

Let the size of the extension field be $q = p^r$, where p is prime. The extension field has generators. Let one of them be α. For any generator α, when we keep multiplying α, we can produce all $q-1$ non-zero elements of the field. And then α is produced again starting cycle. That means that

$$\alpha^q \ mod \ q = \alpha, \ \alpha^{q-1} \ mod \ q = 1.$$

Let

$$x = \alpha^{k_x} \ mod \ q, \ y = \alpha^{k_y} \ mod \ q.$$

Exponent and log are defined as follows

$$exp(k_x) = x, \ log(x) = k_x \ where \ x, k_x \in GF(p^r).$$

Then multiplication of xy is

$$xy \ mod \ q = \alpha^{k_x}\alpha^{k_y} \ mod \ q = \alpha^{k_x+k_y} \ mod \ q$$

$$= \alpha^{k_x+k_y \ mod \ (q-1)} \ mod \ q = exp(k_x + k_y \ mod \ (q-1))$$

$$= exp((log(x) + log(y)) \ mod \ (q-1)),$$

and the inverse of x is

$$\frac{1}{x} = \alpha^{-k_x} = \alpha^{q-1-k_x} = exp(q-1-k_x)$$

$$= exp(q-1-log(x)).$$

Therefore, multiplication involves two log table lookups, one addition, one modulo, and one exponent table lookup. Inverse involves one log table lookup, one subtraction, and one exponent table lookup, making these operations quite efficient. The size of the tables is an important parameter when choosing the size of the extension field: it is 2^q. For current sensor networks, this means that extension fields of size 4 or 8 are good choices, but 16 is probably too large, as the lookup tables will require 64K entries.

VI. Alternative Route

Adding an alternative route in the case of the failure of a given link is yet another way to increase reliability. When a link between two nodes fails, the messages sent through that link will successively be dropped, until the link estimation component is triggered and selects a new route. This process, if prevalent, can eliminate the benefits gained from erasure coding, since many consecutive losses will very likely to be above the tolerance of redundancy added by erasure code. In this case, it should be clear that link-level retransmissions are of no great help, unless used to a prohibitively long extent. A sensible strategy, then, is to detect the failure as soon as possible, and send the packet to an alternative route, if possible.

This points to the need of special support from the routing layer for enabling alternative paths towards the destination. This flexibility ultimately depends on the routing geometry of the routing algorithm [20]. For example, in the case of aggregation, in which nodes route to a parent in the tree to the root, there may be many nodes within communication range that decrease the distance to the root. In geographic routing, there may also be more than one neighbor that allows progress towards the destination. In our evaluation we use an implementation of Beacon Vector Routing (BVR) [12]. We describe the algorithm in some detail in Section VII, but for now it suffices to say that it allows flexibility in the selection of routes. We stress the point that using alternative routes is not particular to BVR, and that our findings in this regard can be reproduced in many other routing disciplines.

Sending packets in alternate routes can be seen as a type of retransmission to a different node, and so in effect increases number of packets injected to the network.

VII. Evaluation

We implemented and evaluated the different reliability options described so far – link level retransmissions, erasure coding, and alternative route – in the context of Beacon Vector Routing. We birefly introduce BVR, and our results follow.

A. Beacon Vector Routing

In our experimental evaluation, we use an implementation of Beacon Vector Routing, a point- to-point routing algorithm for wireless sensor networks. For the purpose of our evaluation, it is not necessary to describe the routing algorithm in much detail, except for its aspects that provide flexibility in selecting routes.

BVR assigns virtual coordinates to nodes, derived solely from the network connectivity information. A subset of r nodes is selected as "beacons", and these beacons flood the network at least once, so that all nodes learn their distance to the set of beacons. The beacons act as reference points for routing. A node p's coordinates are then given by $\mathcal{P}(p) = (B_{1p}, B_{2p}, ..., B_{rp})$, where B_{ip} is the distance between p and B_i. Each node in BVR is required to know its distance to each of the beacons, and the coordinates of its one-hop neighbors.

The basic routing exported by BVR is a route-to-coordinate interface. Routing in BVR is a form of greedy routing, similar to the routing used in geographic routing algorithms. When given a packet to route to a coordinate, a node selects the neighbor whose coordinates are the closest to the destination's coordinates, by some distance metric. The simplest such metric is given by Equation 1 below, and is equal to the sum of the absolute component-wise differences of the two coordinates (a form of an $L1$ metric).

$$\delta(\mathcal{P}(p), \mathcal{P}(q)) = \sum_{i=1}^{r} |B_{ip} - B_{iq}|, \qquad (1)$$

This greedy-routing procedure may fail when no neighbor makes progress in the coordinate space towards the destination. To get out of these 'local minima', BVR employs a fallback routing mode that ultimately guarantees that the destination will be reached. In fallback mode, the node forwards the packet towards the beacon that is closest to the destination. This beacon is readily determined by the smallest component of the destination's coordinates. The minimum distance reached by the packet is recorded in the packet; this allows each node to resume normal greedy routing when one of the neighbors makes progress. Eventually, a packet may reach the beacon which is closest to the destination. In this situation, normal greedy routing cannot be used without the guarantee of no loops. The beacon then initiates a scoped flood that will reach the destination. The choice of the fallback beacon as the closest to the destination minimizes the flood scope.

We can now explain how in BVR one can get flexibility for choosing next hops. At each step of greedy routing, there may be many nodes which make progress in coordinate space to the destination. Also, when doing fallback-mode routing, any node that is reachable and is closer to the desirable root is good to be used. In the BVR implementation, we fix the maximum number of alternative routes to 6, and the routing layer returns these alternatives ordered by progress and link quality.

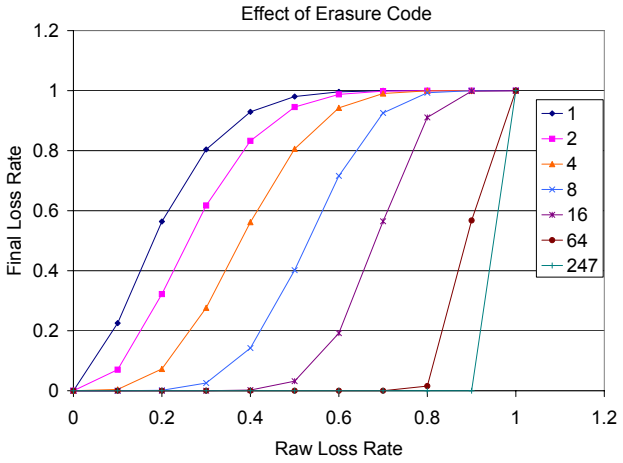

Fig. 7. Increase or decrease in loss rate by using erasure code. Each line indicates how many redundant erasure code words are added to 8 original messages

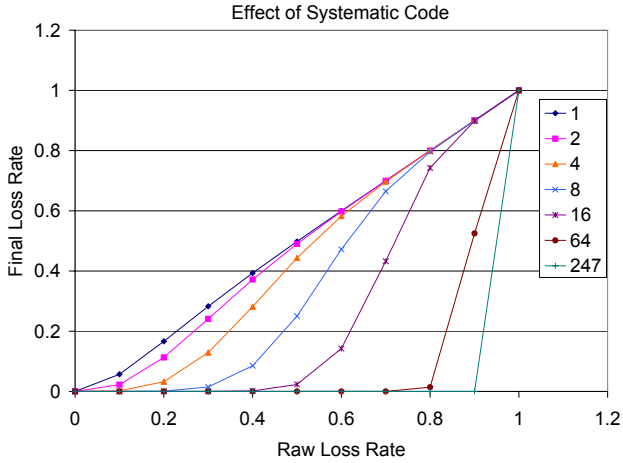

Fig. 8. Decrease in loss rate by systematic code. Each line indicates how many redundant systematic code words are added to 8 original messages

B. Erasure Code

Figure 7 shows how much reliability can be gained from erasure code. This graph is analytically obtained assuming loss is randomly distributed, for different redundancy levels. The bandwidth consumed by additional code words is not shown: more redundant code words achieve higher reliability, but more bandwidth will be consumed also. When there is small amount of redundancy, the loss rate is higher than raw loss rate. This happens because when we can not decode, we loose everything. So receiving 7 packets is effectively same as receiving 0 packets. Systematic code is good not only for saving computation, but also for increasing reliability.

Number of Redundant Code Words	Time (ms)
0	0.780
1	2.539
2	4.298
3	6.057
4	7.816
5	9.575
6	11.334
7	13.093
8	14.852

TABLE I

ENCODING TIME TO PRODUCE ALL CODE WORDS. LEFT COLUMN INDICATES HOW MANY ADDITIONAL CODE WORDS ARE PRODUCED TO 8 ORIGINAL MESSAGES

Number of non-original-message code words	Time (ms)
0	0.427
1	4.027
2	6.876
3	9.820
4	13.713
5	17.119
6	21.059
7	24.604
8	27.065

TABLE II

DECODING TIME OF ALL 8 MESSAGES GIVEN HOW MANY CODE WORDS ARE NOT ORIGINAL MESSAGES

By using a systematic code, even if we receive 7 packets, when 3 packets are codes containing original messages, we get 3 packets. Figure 8 shows the improvement with systematic code. The graph is also obtained analytically. The final loss rate is always smaller than the raw loss rate. All the following tests use systematic code. Later in this section, we examine the trade-off between reliability and bandwidth overhead by varying number of additional code words.

We measured the encoding and decoding speed on MICA2 motes. Messages and code words are 29 bytes long. Message and code words are divided to 8 bit-long units, and there are 8 original messages to send.

Table I shows the encoding time. In systematic codes, the first 8 code words do not require any computation, they are just

Packet Loss Rate	Time(ms)
0	0.427
0.1	3.143
0.2	5.696
0.3	8.263
0.4	10.928
0.5	13.700
0.6	16.548
0.7	19.416
0.8	22.220
0.9	24.832

TABLE III

EFFECT OF LOSS RATE ON TIME TO DECODE 8 MESSAGES

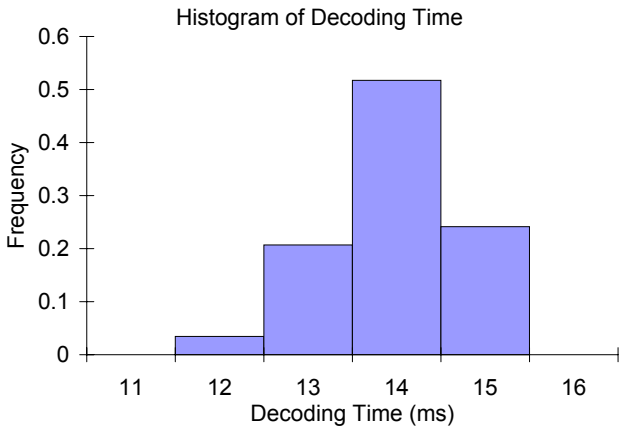

Fig. 9. Histogram of time to decode 8 messages with 4 code words containing original messages

memory copies. Each additional code word requires 1.7ms, which is smaller than the transmission time of a packet (20ms) by an order of magnitude. This means that we can encode on the fly.

Table II shows the decoding time. In systematic code, decoding time depends on the mix of code words. The more code words contain original messages, the quicker the decoding becomes (Section V-B explains it in more detail). Decoding time is roughly linear to the number of non-original-message code words.

Table III shows the expected decoding time calculated from Table II, given the packet loss rate. Decoding time is also roughly proportional to the packet loss rate. Decoding takes less than 30ms even in the worst case, and it takes 160ms to receive the next 8 code words. So each decoding step can be done well before the next decoding occurs, even though there is an issue of buffering that may need to be addressed.

The mix of code words (how many code words are original messages) determines decoding time, but given the the number of code words containing original messages, the combination of code words does not affect decoding time significantly. This is shown in Figure 9. 30 random cases are produced with 4 code words containing original messages with 4 additional code words, in total decoding 8 messages. The average decoding time was 13.44ms, with a 95% confidence interval of 1.52. Standard deviation was 0.74, less than 10%.

Memory usage depends on the size of encoding unit, which is sub code word in code packet as shown in Figure 6. Most of memory requirement comes from operation table and matrix. With 8 bit-long unit, 512 bytes are used for operation table, 64 bytes for matrix, 232 bytes for 8 packet buffers, and 4 bytes are used for other variables. Packet buffer will be provided and shared by application, and operation table can be stored in program memory. The memory usage by the erasure code component is then 68 bytes.

C. Comparing Options

We compared different combinations of options (link-level retransmission, alternative route, erasure code) using experimental data on a real testbed. We ran the case with a maximum of 5 link-level retransmissions, with route fix which tries up to 6 alternative routes (also with 5 maximum number of retransmissions per each next hop). From this data we simulated and computed the results for other cases: 0 to 5 retransmissions per link without route fix. We also compute the results as if the stream of packets consisted of erasure coded packets: we label each packet as being an original packet or a redundant packet, and are able to infer the results of the decoding process by labels of the received packets. Summarizing, in our evaluation we vary two dimensions, retransmission and redundancy. We vary the first from 0 to 5 maximum link-retransmissions with no route fix, and 5 maximum retransmissions with route-fix. We vary the redundancy from 0 to 8 redundant packets for each 8 packets of data. Route fix is only for 5 maximum retransmissions.

The testbed we use is deployed on forth floor of the Computer Science building – Soda Hall – at the University of California, Berkeley. It consists of 78 Mica2Dot motes, deployed in graduate student offices, and is depicted in Figure 10. Test data shown here was collected in the following way: we let the BVR routing information in all nodes stabilize for 75 minutes, and then had an external program send 300 packets of data from one specific node to another. We chose the nodes so that they would be separated a significant number of hops. Packets are separated by 1 second, which is long enough to eliminate interference between two consecutive packets. The pair of nodes considered is also shown in Figure 10.[2] The path we use presented an average of 5 hops across all packets that were delivered, and the overall loss rate in the network was 26.28%. This takes into account all messages that were sent over all links during the course of the experiment.

The metrics we use to analyze the different options are reliability, cost, and overhead. Reliability is the percentage of original data packets that arrive at the final destination. It measures the actual data that two applications at both ends can exchange successfully. Cost is the total number of packets injected into the network in order to transmit one packet of data. Cost includes both effect of loss rate, and the average number of hops from source to destination. Since some options may take a more reliable path even though it could be longer, cost is more meaningful than packet loss rate. However, as we shall see, cost alone also does not tell the whole story, because in the presence of loss one may incur cost and not do useful work. We define overhead as the cost per hop per each successfully delivered data packet. It is normalized by dividing by the path length, allowing us to make more meaningful comparisons. The overhead thus measures the amount of work done in the network (per hop), to deliver one data packet end-to-end. Ideally it should be 1, and we should look for options

[2]We ran other similar experiments among other pairs of nodes with very similar results.

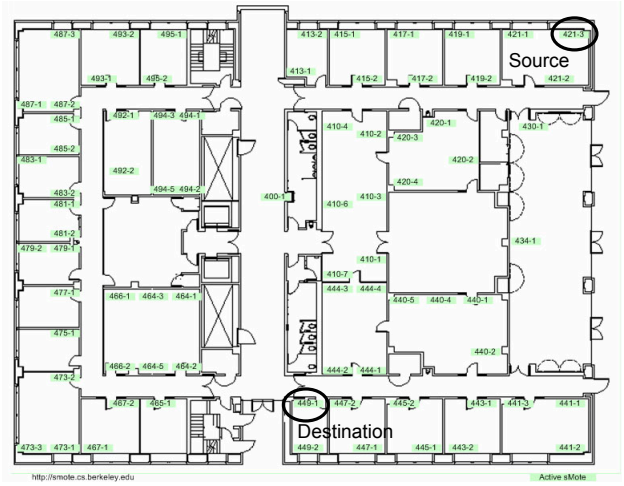

Fig. 10. Map of Soda hall testbed. Source and destination are also indicated.

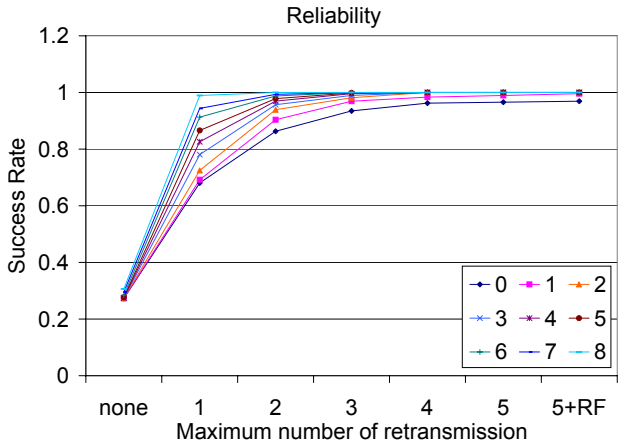

Fig. 11. End-to-end reliability achieved by options. Each line represents number of redundant code words for 8 original messages. RF means route fix is used

that maximize the reliability with the smallest overhead.

Figure 11 shows the reliability each option (link-level retransmission, erasure code, and route fix) achieves. The x-axis shows the number of retransmissions and whether route fix is used. Each curve represents how many redundant code words were added to each 8 original messages. Figure 12 shows the normalized overhead for each option with same x-axis and legends.

Our first observation is that link level retransmissions should be used in any case. With no retransmissions, the reliability is so low that the effect of redundancy is negligible (in spite of adding overhead, as in Figure 12). The low number (less than 30%) seen in Figure 11, is close to the expected for a five hop transmission over links with 26.28% loss rate. When using at most 1 per-link retransmission, not only does the success rate go substantially up, but also does the effect of adding redundancy increase.

Our second observation from Figure 11 is that even with 5 retransmissions and route fix, the reliability does not reach

100%. The reason for this may come from the nature of the loss process: there can be packets which are dropped even after 5 retransmissions, because a link may have gone down, and this information has not yet reached the routing layer or the link estimation component. In these cases, unless some costly measure is taken by the network, that may include holding the packet in buffers for extended periods, or backtracking the packet in the reverse path, it may be inevitable to drop the packet. Erasure codes are useful in this scenario exactly because they do not require that all packets be delivered to recover the data, and it is safe to drop some packets that would otherwise be too costly to deliver.

Erasure codes, however, add a fixed overhead, since redundant packets are always sent at a given rate. We can see in Figure 12 that for a given retransmission option, the overhead always increases with the number of redundant packets. With little redundancy added, the overhead, as shown in Figure 12, decreases as retransmission increases. This is mostly due to the increase in the success rate, and thus the decrease in wasted effort to deliver packets. We can notice, however, that with 4 or more redundant packets per each 8 data packets (50% or more redundancy added), the overhead increases with more retransmissions, with no corresponding gain in reliability. With high redundancy, destination already gets enough number of code words to reconstruct original data. Additional packets delivered by more retransmissions do not increase reliability any more. They just add cost to the network. Also, when maximum number of retransmissions is large and end-to-end reliability is high, erasure code wastes too much bandwidth, and overhead gets high.

Finally, in Figure 13 we plot, for the different options, reliability versus overhead. These plots give insight into the tradeoff at hand, and we caution the reader that the axis have different roles than in previous plots. Each curve in the figure corresponds to one retransmission option, and the nine points in each curve correspond to the redundancy. In all curves redundancy increases from left to right. In this graph, we would like to choose points that have overhead close to 1, and reliability close to 1, thus as close to the upper left corner as possible. We can notice that adding on-demand retransmissions increases the reliability without incurring in overhead, at least for low redundancy cases. On the other hand, adding redundancy, while always incurring overhead, is needed to get the last few percent of reliability. We see that for a given retransmission option, in order to add reliability one has to add redundancy, but the gains are very different for different maximum number of retransmissions.

In table IV, we pose the question of how one chooses an option, given these trends in reliability and overhead. As the threshold increases, sweet spot moves toward more redundancy. And when the number of redundant code words increases, maximum number of retransmissions drops. When the number of redundant code words increases, more packet losses can be tolerated, so retransmission for additional packet delivery becomes not necessary.

Causes of failures are shown in Table V. Data is from case

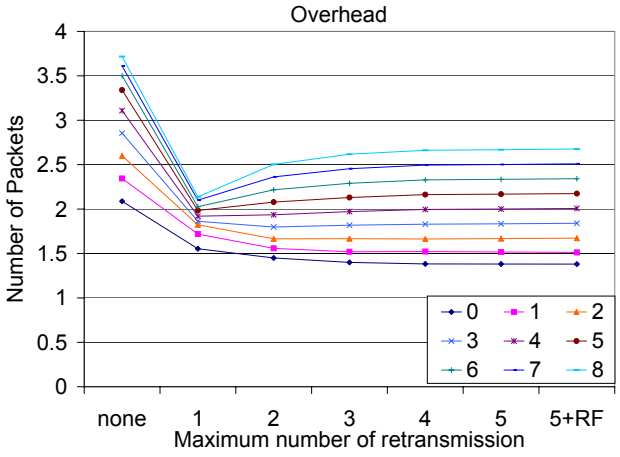

Fig. 12. Number of packets injected to network per hop per successfully received data. Each line represents number of redundant code words for 8 original messages. RF means route fix is used

Threshold	Retransmission	Redundancy	Overhead
90%	5+RF	0	1.381
95%	5+RF	0	1.381
98%	5+RF	1	1.512
99%	5+RF	1	1.512
99.9%	4	2	1.663

TABLE IV

GIVEN A THRESHOLD RELIABILITY REQUIREMENT, WHAT IS THE RETRANSMISSION/REDUNDANCY COMBINATION THAT HAS THE SMALLEST OVERHEAD?

with 5 maximum retransmissions and route fix, and graph is simulation for case with 5 maximum retransmissions without route fix. It is to show the effectiveness of route fix. 'Reroute' is failure without route fix, but which succeed with route fix. 'Nowhere to send' is failure without route fix, and also failure with route fix: it could not send to any next hop candidates. This failure happens when a packet can be delivered into node, but cant be forwarded out. 'Queue Overflow' happens when pending outgoing packets fill up queue, and new packet arrives. Reroute and queue overflow are divided to independent failure and consecutive failure. Consecutive reroute failure indicates stale routing table value. Beacon vector routing adapts to link failure quickly, and we did not shoot packets too quickly in the test, so there is no stale routing table problem. Queue overflow constitutes 80% of failure. Congestion control needs to come in. When link-level retransmission and route fix are used, packets tend to reside in queue longer until they are successfully delivered to the next hop. Then it increases chance of queue overflow. Therefore, those options may not always increase reliability.

VIII. FUTURE WORK

This work presents an initial evaluation of several options for achieving reliability. We leave as subsequent work further exploration of the design space. Route fix is tested only with 5 maximum retransmissions, which already provides high

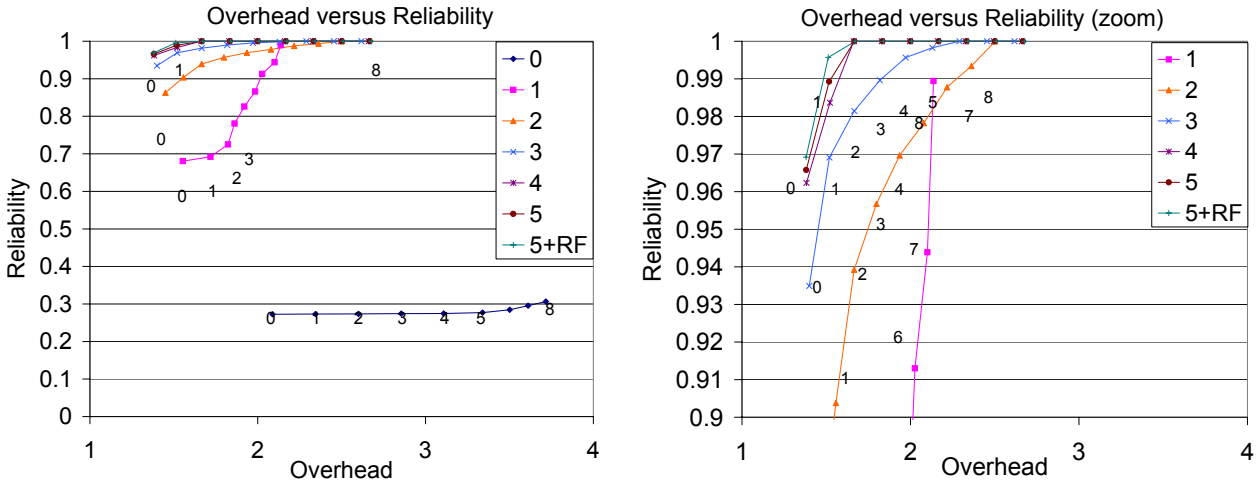

Fig. 13. Overhead versus reliability for different combinations of retransmission and redundancy options. Overhead is the number of packets injected per hop per received data packet. Points in the same curve have the same retransmission option, and each curve has 9 points (indicated by numbers), corresponding to the number of redundant packets for each 8 packets of data.

Cause	Percentage
Independent Queue Overflow	2.667%
Consecutive Queue Overflow	0%
Independent Reroute	0.333%
Consecutive Reroute	0%
Nowhere to send	0.333%

TABLE V
DECOMPOSING CAUSES OF FAILURES

reliability. Even though we see marginal improvement at this point, it surely not only improves reliability but also decreases the overhead. It will be interesting to see case with route fix and small maximum number of retransmissions, and compare cost per reliability to case with no route fix and large maximum number of retransmissions.

A direct comparison with end-to-end retransmission is missing. For very high reliability end-to-end retransmission would be attractive solution, even though it will increase delay.

Thick path, in which messages are forwarded simultaneously by several nodes that make progress towards the destination is another possible option. It achieves reliability only through information redundancy, and can survive link failure. Moreover it has low delay to deliver packet. The downside is that it injects a large number of packets: it is multiplication of path length and path thickness. Since traffic is correlated locally, channel contention will not significantly affect whole network. However, in terms of energy consumption this would be a bad choice. It will be interesting to see trade-off of success rate, overhead, delay, and energy consumption.

Some form of congestion control is needed. Large chunk transfer and admission control would be a good candidate solution. This enables back pressure working as congestion control without much overhead.

Initially it looked like that our implementation of erasure codes worked as long as $M + N < 2^r - 1$. In experiments, when $M > r$ it worked in most cases, but not always. Mathematical reasoning of this phenomenon is also left for future work. And if we can avoid these cases without expensive operation, it would be helpful. If we can have larger M, tolerating any 6 packet losses provides more robustness than tolerating 3 packet losses from each of two transfers.

IX. Conclusion

Diverse options for achieving reliable transfer in wireless sensor networks are discussed, implemented, and tested in a real testbed. Link-level retransmission, erasure code, and route fix are implemented and evaluated. Link-level retransmission handles transient link failure and contention very efficiently. Erasure code introduces static overhead, however its use loosens the burden of delivering the last few packets (99.99% versus 99%), which are very expensive and inefficient using other methods. Route fix solves stale routing table problem, providing quick adaptation to link failure, if the routing layer provides flexibility in route selection. In turn, route fix reduces consecutive losses, increasing usefulness of erasure code, which does not work well with successive losses. Link-level retransmission happens on demand: packets are retransmitted only when necessary. Route fix is also on-demand: only when packet can not be forwarded to next hop. Those local and on-demand options are very efficient approaches (cost per reliability). Erasure code allows some flexibility in the losses, and route fix provides flexibility in selecting next hop. Some options address some problems efficiently but not all failures, which can be effectively cured by some other options. Our results show that combining options would provide a sweet spot.

X. Acknowledgments

Kevin Fall gave invaluable comments on analyzing option space. This work was supported by the Defense Advanced Research Projects Agency (grant F33615-01-C-1895), the National Science Foundation (grants EIA-0122599 and IIS-033017), the Department of Energy (grant DR-03-01), the

Center for Information Technology Research in the Interest of Society (CITRIS), Intel, Sun Microsystems, Hewlett Packard, and Microsoft.

REFERENCES

[1] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proceedings of the First International Conference on Embedded Network Sensor Systems*, 2003.

[2] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," *ACM Sensys*, November 2003.

[3] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[4] V. Jacobson and M. J. Karels, "Congestion avoidance and control," in *In Proceedings of the Sigcomm '88 Symposium*, Stanford, CA, Aug. 1988.

[5] M. Maroti, "Directed flood-routing framework," ISIS, Vanderbuilt University, Tech. Rep. ISIS-04-502, 2004.

[6] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks," in *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[7] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys'04)*, 2004, to appear.

[8] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2002.

[9] D. Ganesan, "TinyDiffusion Application Programmer's Interface API 0.1," http://www.isi.edu/scadds/papers/tinydiffusion-v0.1.pdf.

[10] M. D. Yarvis, W. S. Conner, L. Krishnamurthy, A. Mainwaring, J. Chhabra, , and B. Elliott, "Real-world experiences with an interactive ad hoc sensor network," in *Proceedings of the International Conference on Parallel Processing Workshop*, 2002.

[11] Y. Kim, R. Govindan, B. Karp, , and S. Shenker, "Practical and robust geographic routing in wireless networks," Under submission.

[12] R. Fonesca, D. Culler, S. Ratnasamy, S. Shenker, and I. Stoica, "Beacon vector routing: Scalable point-to-point routing in wireless sensornets," In submission.

[13] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Psfq: a reliable transport protocol for wireless sensor networks," in *Proc. of the 1st ACM international workshop on Wireless sensor networks and applications*. ACM Press, 2002.

[14] F. Stann and J. Heidemann, "Rmst: Reliable data transport in sensor networks," in *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*. IEEE, April 2003, pp. 102–112.

[15] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, April 1997.

[16] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An xor-based erasure-resilient coding scheme," International Computer Science Institute, Tech. Rep. TR-95-048, 1995.

[17] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of the ACM SIGCOMM '98*. ACM Press, 1998.

[18] P. Maymounkov, "Online codes," *NYU, Technical Report TR2002-833*, November 2002.

[19] "http://mathworld.wolfram.com/."

[20] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of dht routing geometry on resilience and proximity," *ACM SIGCOMM*, August 2003.