On Improving Wireless Broadcast Reliability of Sensor Networks Using Erasure Codes

Rajnish Kumar[1], Arnab Paul[2], Umakishore Ramachandran[1], and David Kotz[3]

[1] College of Computing, Georgia Tech, Atlanta, GA 30332
[2] Intel Corp., Hillsboro, OR 97124
[3] Computer Science Depratment, Dartmouth College, Hanover, NH 03755

Abstract. Efficient and reliable dissemination of information over a large area is a critical ability of a sensor network for various reasons such as software updates and transferring large data objects (e.g., surveillance images). Thus efficiency of wireless broadcast is an important aspect of sensor network deployment. In this paper, we study FBcast, a new broadcast protocol based on the principles of modern erasure codes. We show that our approach provides high reliability, often considered critical for disseminating codes. In addition FBcast offers limited data confidentiality. For a large network, where every node may not be reachable by the source, we extend FBcast with the idea of repeaters to improve reliable coverage. Simulation results on TOSSIM show that FBcast offers higher reliability with lower number of retransmissions than traditional broadcasts.

1 Introduction

We consider the problem of information dissemination in wireless sensor networks (WSN). This is an important domain of research because of the multitude of potential applications such as surveillance, tracking and monitoring. WSN nodes are resource constrained, and thus they are initially programmed with minimal software code, and are updated whenever needed. For such on-demand programming, broadcast is typically used to disseminate the new software, making broadcast-efficiency a very important aspect of WSN deployment.

An efficient wireless broadcast scheme must solve two key interrelated challenges:

(*i*) *Messaging Overhead:* Traditionally, each node of a WSN rebroadcasts any new data packet, resulting in too many unnecessary transmissions [14]. For example, if a software update of k packets is to be sent over a WSN of n nodes, potentially k times n broadcasts could be sent out. The larger the number of broadcasts, the more cumulative power is consumed because of the communication. Furthermore, the increased messaging overhead introduces more collisions and thus affects the channel reliability.

(*ii*) *Reliability:* The reliability of message dissemination is a key requirement for a sensor network to function properly. For example, in case of a global software update, if the software at all nodes are not updated reliably, the collected data may become erroneous, or the network may run into inconsistent state. To avoid such problems, reliable code dissemination becomes important. But, empirical results establish that

J. Cao et al. (Eds.): MSN 2006, LNCS 4325, pp. 155–170, 2006.

wireless channels are often lossy [2], and in presence of channel loss and collisions, achieving high reliability becomes difficult.

So far, two baseline approaches have been proposed in the literature, *viz.,* deterministic and probabilistic flooding [13]. It turns out that simple deterministic flooding protocols are quite inefficient to address the issues mentioned above. In a probabilistic approach, each node randomly decides whether or not to broadcast a newly seen data packet. These baseline approaches do not assume any extra information about the networks. Several variants and optimizations over these two baseline schemes have also been introduced [14,16,5]. Typically, these derivatives either assume some structural information about the networks, e.g., the knowledge of the network neighborhood, internode distances, views on the possible local clusters and so on, or, the protocols rely upon additional rounds of messages, such as periodic ' 'Hello" packets, and ACK/NACK packets following every broadcast.

However, it may not often be possible to depend on any additional information for reasons that are specific to sensor networks. For example, the nodes may not be equipped with GPS, or deployed in an area with very weak GPS signals. The information on neighborhood, distance, location, etc., may continue to change due to mobility and failures. The periodic ' 'gossip" becomes expensive to support because of the transmission overhead incurred and dynamic nature of WSN.

Instead of a protocol that relies completely on controlling the communication, our intuition is to aid the messaging with computational pre/post-processing. The emerging hardware trend suggests that future sensors would have significant computing power. For example, devices such as an iMote have up to 64 KB of main memory and can operate at a speed of 12 MHz. Extrapolating into the future, the motes will soon possess as much computing resources as today's iPAQs. However, while processor efficiency (speed, power, miniaturization) continues to improve, networking performance over the wireless is not expected to grow equally, merely because of the physical ambient noise that must be dealt with. Thus trading processor cycles for communication can offer many-in-one benefits in terms of smaller messaging overhead, less collision, enhanced reliability and reduced power consumption. Following this intuition, we propose a new baseline protocol that is based on a fundamentally different principle, that of the forward error correcting codes (FEC). [1]

The contributions and the findings of this paper can be summarized as follows:

(*i*) We present a new design principle for wireless broadcast in sensor networks. The idea is to combine erasure coding with probabilistic broadcast technique. Founded on this FEC principle, the new WSN broadcast protocol, FBcast, offers high reliability at low messaging overhead. The new scheme also provides additional confidentiality. FEC has earlier been used for asynchronous data delivery and IP multicast in wired networks, but to the best of our knowledge, ours is the first work to explore the viability of applying FEC in wireless sensor networks that have unique requirements and packet loss characteristics substantially different from wired networks. Ours is a vanilla data

[1] Erasure codes are a class of encoding; a data packet (seen as a collection of small blocks) is blown up with additional redundant blocks (such as parity checksums) so that if some blocks are lost due to any kind of noise (external signal, faults, compromises), the original packet may still be reconstructed from the rest.

dissemination protocol that assumes no extra information about the underlying network. As we observe through our experiments, the transmission characteristics (such as signal strength and packet loss) vary fairly randomly as one goes radially outward from a data source; thus common assumptions, such as regular signal strength distribution over concentric circles or sphere, that are made by many of the other protocols do not hold true in reality. Using FEC based vanilla protocols in such a scenario becomes quite useful.

(*ii*) We compare FBcast with probabilistic broadcast through simulation studies using the TOSSIM simulator [4]. Protocol efficiency can be evaluated over more than one axis, each of which can be potentially traded for another, e.g., reliability can be enhanced at the cost of extra messaging overhead, or spatial density of the sensors and so on. Thus a point-by-point fair comparison between these approaches may not always be possible. However, our experiments do suggest that, FBcast performs better over a larger parameter space formed by the metric-axes.

(*iii*) We propose FBcast with *repeaters* to disseminate code over large area network. Over a multi-hop network, especially in sparse network deployment, traditional broadcast reliability decreases as one goes away from the data source. We extend the basic FBcast protocol with the idea of repeater nodes. Where to place the repeaters without much network information is a challenge. We present a novel heuristic to solve the repeater placement problem. We compare the performance of the extended FBcast protocol against a similar variant of probabilistic protocol, and find the new protocol more effective in increasing the broadcast coverage.

The paper is organized in the following way. Section 2 looks at the motivation and related broadcast protocols to place our work in context. Section 3 provides details of FBcast protocol. It also explains the encoding scheme used. Section 4 presents the implementation details of FBcast and simulation results.

2 Background and Related Work

Baseline Approaches: So far, the data dissemination techniques for wireless networks can be divided into two major approaches, deterministic or probabilistic broadcasts. Simple flooding [10] is the most naive implementation of the first class. However, naively re-broadcasting can easily lead to broadcast storm problem [14], and hence the need for controlled density-aware flooding and multicast algorithms for wireless networks [1]. Simple flooding (depending on the placement of neighboring nodes) also suffers from the severe inefficiency that the effective additional coverage of a new broadcast can be as low as 19% of the area of the original circle that a broadcast can effectively reach.

Variants and Optimizations: Several other optimizations can be applied to these baseline schemes. For example, *pruning* is the strategy of selectively suppressing the broadcast activity. The objective here is to find out exactly a set of nodes (that will broadcast data) so that no other node need to rebroadcast. These nodes constitute a *Flooding Tree*. Finding a minimal flooding tree is NP-complete [6]. The *Dominant Pruning* (DP) algorithm is an approximation to finding the minimal flooding tree [6]. Lou and Wu proposed further improvements over DP that utilize two-hop neighbor information more

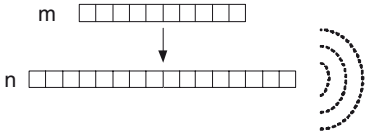


Fig. 1. FBcast at source: m original packets are encoded into n packets and injected in the network

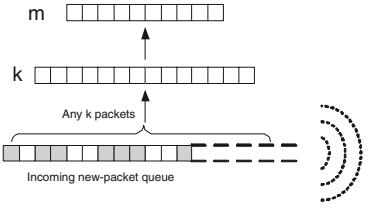


Fig. 2. FBcast at recipients: k packets are chosen from received queue and decoded back to m

effectively than DP [7]. Again, the neighborhood information is maintained by periodic gossiping that add additional transport overhead. Garuda [12] provides reliable downstream data broadcast using a minimum dominating set of core nodes, which provides a loss recovery infrastructure for remaining nodes. The overhead incurred by core selection and maintenance in Garuda may make it an expensive solution for dynamic networks.

Similarly, since collision is a critical obstacle, one intuition is to have many of the nodes stay off from transmissions, and thus create a virtual sparser topology that will have less collisions. Irrigator protocol and its variants are based on this idea [11]. For a comparative and comprehensive survey of these protocols, the reader can refer to a related work[16]. PSFQ [15] uses hop-to-hop error recovery by injecting data packets slowly, while neighbor nodes use NACK-based fast repair. On a similar note, Trickle [3] combines epidemic algorithms and density-aware broadcast towards code dissemination in WSN.

FBcast, as a base approach, provides another alternative to simple deterministic and probabilistic broadcasts. Other smart adaptations such as location-aware retransmission, maintaining neighborhood and routing information are expected to boost its performance. Rate-less erasure codes, such as Fountain codes, form a critical ingredient of our approach. They were first proposed as a tool for efficient multicasting. Later on, versions of such codes have been shown as useful tool for Peer to Peer file sharing and download purposes [9]. FBcast applies the idea of fountain encoding with the previously known scheme of probabilistic gossip, to achieve high reliability without the extra messaging overhead, in the domain of wireless networking where bandwidth, power and reliability are very critical issues to be addressed.

3 FBcast Protocol

Figure 1 and 2 pictorially represent FBcast broadcast protocol. The data to be disseminated consists of m packets. The source blows up these m packets into n packets, using the encoding scheme described below, and the encoded data is injected in the network. Each recipient rebroadcasts a packet, if it is new, with a probability p. When a recipient node has received enough data packets ($k \geq m$). The exact value of k depends on the specific encoding scheme used. for decoding, it reconstructs the data and passes it

to the application. In order to encode and decode (using FEC) the nodes would need to know a random seed from which the rest of the code parameters can be generated using a pseudo random number generator. We assume that this seed and the generator (typically a very light-weight algorithm) is shared by all nodes.

Erasure codes provide the type of encoding that is needed to accomplish the protocol. In particular, it is desirable that the codes have following properties. (*i*) The ratio n/m (also known as the *stretch factor*) can be made arbitrarily large or small flexibly. In other words, one can generate as many redundant packets as needed, by a decision that can be taken online. (*ii*) There is no restriction on the packet length. (*iii*) Both encoding and decoding should be inexpensive.

Standard erasure codes, such as the Reed-Solomon codes, are *inflexible* in the first two aspects, and are quite inefficient in performance. Although these codes allow any desired stretch factor, this can only be done statically. It is not easy to change n and k *on the fly* during the application runtime for the following reasons. First, these codes require that every time the stretch factor is to be readjusted, a new code needs to be defined and disseminated to all the participating nodes. Second, the code length parameter n is always upper bounded by the order q of the underlying field; every time a higher stretch factor needs to be applied, a great deal of meta-information needs to be disseminated and computational overhead incurred. Third, the size of a symbol, i.e., the number of bytes treated as one unit of information, is also upper bounded by the field size; for a field size q, the largest unit of information treated at one time can be at most $\log q$ bits. In our setting, the size of one packet is essentially the symbol length of the code being used, and thus essentially the chunk of data that can be handled at one time is limited by this *a priori* fixed parameter. For a comprehensive discussion regarding the kind of problems posed by standard codes, the reader can refer to the works of Luby, Mitzenmatcher *et al.* [8].

Fortunately, a modern class of erasure codes solves these problems effectively. These are called *Fountain codes*. The category-name *fountain* is suggestive - when one needs to fill up a cup of water from a fountain, there is no need to bother about which particular droplets are being collected, rather just enough number of drops to fill in the glass would be sufficient. Not all Fountain codes are equally flexible. The candidate ones that we are particularly interested in are the Luby Transform codes (LT codes [8]), Raptor and Online codes. We are interested in the codes that are *rateless*, *i.e.*, can produce on-the-fly a *fountain* of encoded blocks from k original message blocks. For a pre-decided small number ϵ, only $k = (1 + \epsilon)m$ number of data blocks out of this fountain suffice to reconstruct the original document. Moreover, there is no limitation on the symbol-size, i.e., the packet length - any number of bytes can be considered as a single unit of information. An example of a rate-less code is the Luby-Transform codes [8]. Our idea is to generate the blocks and sprinkle them over to the neighbors who would re-sprinkle a small fraction of the fountain.

The main benefit of data encoding is three fold. (*i*) Enhanced reliability, which is achieved by adding extra information encoded in the data packets. Thus, if a node has noisy reception, it may not be able to receive all the data packets, yet, it can generate the original data. (*ii*) Data encoding decreases transmission overhead. Because of the redundancy, the recipients do not need to retransmit all the packets; each transmits

only a few of what it receives, thus alleviating contention for the shared channel. (*iii*) The scheme provides data confidentiality as an extra benefit. Because of the shared nature of the wireless channel, confidentiality is often a requirement in wireless network deployment. To encode and subsequently decode the same data, the sender and receiver need to have a shared random seed. Hence, no eavesdropper can decode the information from the wireless channel.

4 FBcast Evaluation

We have implemented the communication aspect of FBcast protocol in TinyOS, i.e., we account for only packet transmission and reception. Though we do not implement the data encoding and decoding in TinyOS, we utilize our experience of fountain code implementation, discussed below, on Linux to tune the FBcast parameter (stretch factor). While we explore the effect of encoding/decoding control parameters upon FBcast reliability, we do not evaluate their effect on the energy consumption or computational latency they add to the broadcast because of the focus of this paper. To understand the protocol behavior, we simulated FBcast using TOSSIM [4] for different network sizes. For comparative study, we also implemented traditional probabilistic broadcast, *pbcast* in TinyOS.

We have looked at three aspects of FBcast and *pbcast*: reliability, transmission overhead, and latency. Reliability is measured as the percentage of motes that receive the original message being disseminated. If a mote receives only some of the injected packets, it may not be able to reconstruct the original data; we assume this to be true for both FBcast and *pbcast*. Transmission overhead is the sum total of transmitted packets on all the nodes during the simulation time. The simulation time is kept long enough such that retransmissions by every mote is complete. Latency is the average time when motes are able to reconstruct original data after receiving enough packets, and it does not include the data encoding or decoding time. For FBcast, latency is the expected time when motes have received k packets, and for *pbcast* it is the expected time when motes have received all the injected packets.

The FBcast parameters are set as follows: $m = 10, n \in \{20, 40, 60\}, k = 14$, and p is adjusted in proportion to n. More precisely, p varies from $1/n$ to $8/n$, thus, for n=20, p is varied from 0.1 to 0.4. Putting this in words, the number of packets in the original data is 10. With a stretch factor of 2, FBcast encodes the data to 20 packets and injects them at the source. Our experiments reveal that a factor of 1.4 is sufficient, i.e., a mote that receives at least 14 distinct packets, can reconstruct the original data. In case of simple broadcast, only 10 packets are injected. For FBcast, value of p is kept proportionally low as n is varied. For $n = 20$ and $p = 0.4$, a mote is expected to retransmit 8 out of 20 new packets it receives. The retransmission overhead here thus becomes equivalent to that of *pbcast* with $p = 0.8$. For *pbcast* experiments, due to absence of any encoding or decoding, $n = m$.

A few words about the implementation of Fountain code. Our experience of implementing fountain codes suggests that by choosing $m' \approx 1000$ (number of message symbols) and $n' \approx 6000$ (number of encoded symbols), data can be reliably reconstructed from $k' \approx 1400$ symbols. However, a bunch of symbols can be coalesced together to form a packet, e.g., by coalescing 100 symbols one generates message blocks of size

$m = 10$ packets and encoded blocks of size $n = 60$ packets. The memory requirement is also within the limits of present motes. For example, to implement LT codes (a class of fountain codes), one needs to have in memory a bipartite graph of size $n' \times \log(m/\delta)$ (see Theorem 13 in [8]). δ is a small constant (e.g., $\delta = 10^{-3}$ gives us very high accuracy in the decoding). Thus, for the parameter set we have presented in this paper, and the most space-efficient representation of the data structures, the memory requirements would be a little over 60 KB, which is clearly not far from the current limits. Moreover, it is expected that the main memory will soon touch the limits of megabytes, thus paving for more time-efficient representations of the structures for these algorithms. In our TOSSIM experiments, we simulated a network of mica2 motes. These motes presently have only 8 Kilobytes of memory, not enough for a full-scale implementation. However, devices such as iMotes already have 64 KB memory, and it is only fair to assume that very soon, enough space will be available on these motes for running both OS and the applications of this order.

Results Summary. FBcast and *pbcast* both can achieve reliability close to 100%, but FBcast can do so for larger window of variation in the mote density, and at lower transmission overhead than *pbcast*. Also, while *pbcast* exposes only the forwarding probability to control its behavior, FBcast exposes the forwarding probability and the stretch factor as control. Thus FBcast can be adapted more flexibly to suit different deployment densities and reliability requirements. The repeater variants of *pbcast* and FBcast, designed for large network deployments, both have higher reliability compared to their original counterparts. However, FBcast variant is easier to configure and it attains more than 99% reliability for various deployment parameters at lower transmission overhead compared to the $pbcast$ variant.

The rest of this section is organized as follows. First, after explaining the network model used, we start with simple experiments, where there is no rebroadcast, and observe the possible benefits of using FEC. Then we add probabilistic retransmissions to increase the reliability and increase the broadcast coverage. We also explore different ways in which FBcast can be configured. Finally, we add the idea of repeaters in FBcast to overcome the limitation observed for FBcast without repeaters, namely, broadcasting over a very large area.

4.1 Network Model and Assumptions

Unless specified otherwise, the following experiments are based on the empirical radio model supported in TOSSIM, where every link is used as a lossy channel with loss probability based on empirical data. Instead of a perfect radio model, we use the empirical radio model because it allows us to see the effect of packet loss upon broadcast. TOSSIM provides a Java tool, $LossyBuilder$, for generating loss rates from physical topologies. The tool models loss rates observed empirically in an experiment performed by Woo et al. on a TinyOS network [2]. LossyBuilder assumes each mote has a transmission radius of 50 feet. Thus, each mote transmits its signal to all nodes within 50 feet radius range, and the quality of received signal decreases with the increase in distance from the transmitter. Given a physical topology, the tool generates packet loss rates for each pair based on the inter-mote distance.

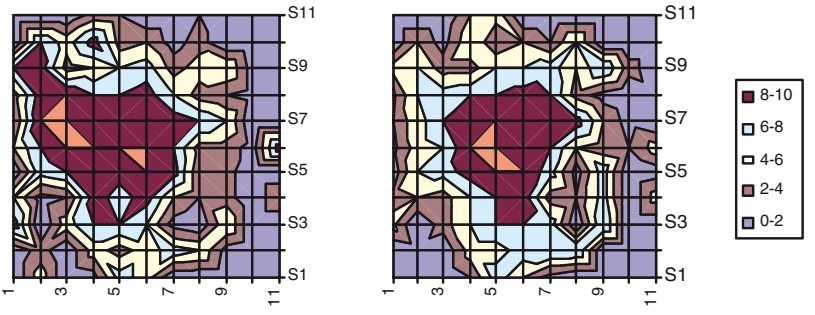


Fig. 3. Topographical picture of reliability for two typical runs showing how the reliability decreases as we move away from the grid center. 121 motes placed on 11x11 grid with inter-mote spacing of 5 feet.

For experiments that use the empirical radio model, we use a grid-based topology to get the loss pattern. By varying the grid size, inter-mote distance is varied, thus affecting the loss probability. The data source is assumed to be at the grid center because of the nature of the experiments. For experiments that use the simple radio model, the transmission loss probability between any two motes is the same for all mote pairs. Nodes are assumed to be located such that each node can listen to all the other nodes. In TOSSIM, network signals are modelled such that distance does not affect their strength, thus making interference in TOSSIM generally worse than the expected real world behavior. However, due to the TinyOS CSMA protocol, the probability of two motes, within a single cell, transmitting at the same time is very very low.

4.2 FBcast Without Any Rebroadcast

We distinguish between rebroadcast and retransmission that we will maintain throughout the rest of the discussion. Whenever a node transmits the same message that it has transmitted in the past, we refer to the event as a *retransmission*. However, when a node is broadcasting a message that is received from another node, the event is called a *rebroadcast*.

We first consider the case when a single source broadcasts a message and there is no other rebroadcast following this event. However the source may retransmit the message multiple times. Reliability is defined as the fraction of nodes that are able to receive (reconstruct) the original message. Thus, reliability depends on packet loss, which in turn depends on multiple factors, including the bit-error rate and interference at the receiving node. Since there is no rebroadcast, there will be no interference. Results (omitted because of lack of space) show that using FEC improves reliability compared to simply re-injecting the original packets multiple times, but without any extra mechanism, or rebroadcasts, it does not provide enough reliability.

If we look at the number of packets being received at different motes, because of the probabilistic nature of channel error, the resulting topological distribution pattern for successful reception is quite dynamic across different simulation runs. Topologies obtained for two typical runs are shown in Figure 3. There exists a set of concentric

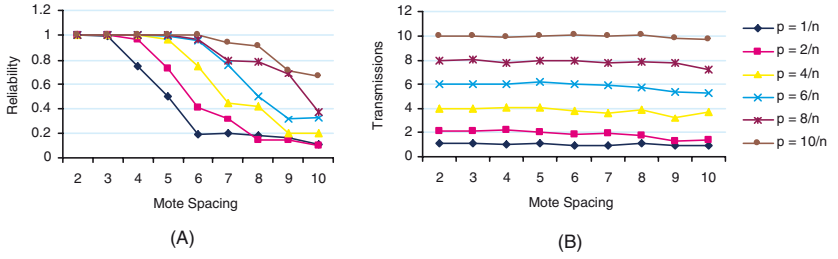


Fig. 4. Pbcast performance for 121 motes deployed on a 11x11 grid with varying inter-mote spacing (x-axis)

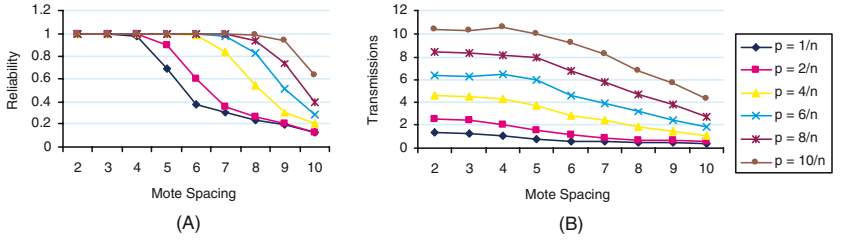


Fig. 5. FBcast (with probabilistic rebroadcast) performance for 121 motes deployed on a 11x11 grid with varying inter-mote spacing (x-axis)

bands of motes that receive similar number of packets, but the bands are not circular, nor are they identical across different runs. The bands are neither circular nor repeatable because of the nature of wireless medium and mutual interferences. This means there is no simple way in which we can divide a large area into smaller cells and put a broadcast server into each cell to provide reliability in a large area. Hence, we resort to another intuitive alternative to increase reliability, i.e., by doing a probabilistic rebroadcast at intermediate motes.

4.3 FBcast with Probabilistic Rebroadcast

When a node is allowed to do probabilistic rebroadcast of new packets, the results show that for the same deployment of 121 motes as before, we can do reliable broadcast to all the motes even at higher inter-mote spacing (than mere 2 feet) is achievable by increasing the forwarding probability at intermediate motes. Both Pbcast (probabilistic broadcast variance without FEC) and FBcast (FEC variant) achieve complete reliability, but as shown in Figures 4 and 5, FBcast achieves higher reliability (Figure 4-A and 5-A) than Pbcast at lower transmission overhead (Figure 4-B and 5-B).

Let us represent the forwarding probability p as α/n, where α is the number of forwarded packets, and n is the original number of packets. At first glance, it may appear that for a given α, say $p = 10/n$, FBcast always gives higher reliability than Pbcast. But if we look closely, we find that there is no direct correlation in the reliability of Pbcast and FBcast for the same α in p. For example, with $p = 10/n$, and spacing

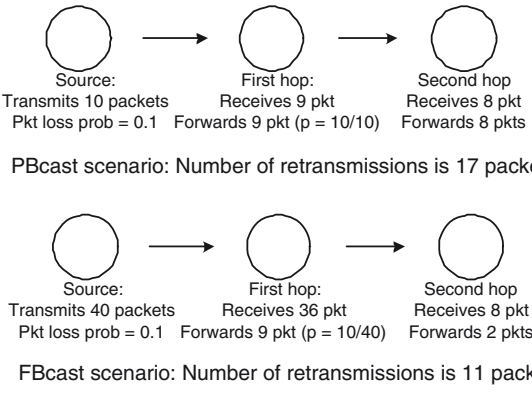


Fig. 6. Transmission overhead comparison of Pbcast and FBcast for a simple topology

of 10 feet, Pbcast has better reliability than FBcast, but at spacing of 8 feet, FBcast gives better reliability. This can be explained if we look at the transmission overheads of Pbcast and FBcast that are shown in Figures 4-B and 5-B.

Contrary to intuition, we see that at $p = 10/n$, at spacing of 10 feet, Pbcast transmits about 10 packets per mote, while FBcast transmits only about 4 packets per mote. We expected both Pbcast and FBcast to incur same amount of transmissions, i.e., about 10 packets per mote (p being $10/n$). The amount of transmissions explain why FBcast has lower reliability than Pbcast. But to understand why FBcast has lower transmission than Pbcast for the same α, we can look at a simple model shown in Figure 6. The three motes, placed in a straight line with one-hop spacing, incur only 11 retransmissions in the case of FBcast, compared to 17 extra transmissions (due to rebroadcast) in the case of Pbcast. This is because though we limit the number of extra transmissions by changing α, the amount of new packets received at distant hops is not proportional to n for Pbcast and FBcast, thus FBcast does fewer transmissions due to rebroadcasting. Also, we have observed that at higher inter-mote spacing, the number of transmissions for Pbcast decreases and the curve becomes similar to what we show in the case of FBcast.

From the above results, we learn that FBcast can provide higher reliability than Pbcast for similar amount of retransmissions; but it may not necessarily mean that FBcast will provide higher reliability than Pbcast for same α. Still, the reliability is limited at higher inter-mote spacing. How much is it possible to stretch the reliability by increasing FBcast's stretch factor? To answer this, we look into the results shown in Figure 7.

For a deployment of 121 motes with 10 feet inter-mote spacing, we can achieve close to 100% reliability at higher stretch factors (e.g. 6) and at high forwarding probability, shown in Figure 7. As expected, increasing stretch factor improves the reliability and also increases the number of retransmissions. Also, for same the stretch factor, increasing the forwarding probability improves the reliability. However, for stretch factor of 2 ($n = 20$), we observe that reliability first improves, peaks at $p = 16/20$, and then it goes down rapidly. To understand this anomaly, we look at the effect of the rate of broadcast at the data source.

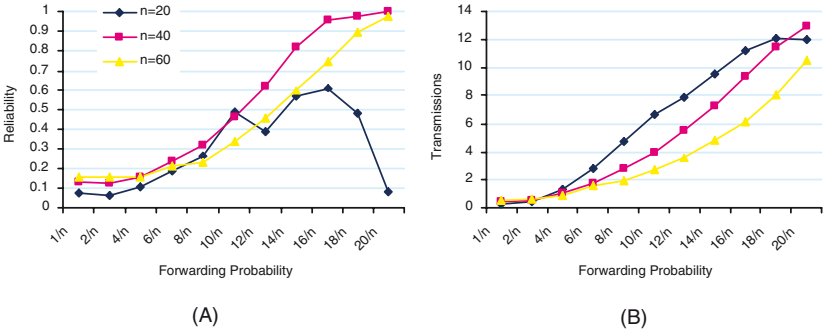


Fig. 7. Effect of stretch factor on reliability. Mote spacing = 10 feet. 121 motes deployed on a 11x11 grid. Forwarding probability is varied along x-axis.

Consider two extreme cases: first, when the source packets are inserted into the network at a very slow rate, and second, where data packets are being inserted without much delay between the transmissions. At a slow rate of source broadcast, there is less interference, and thus higher reliability, compared to the case where data packets are being inserted rapidly. The interference becomes more severe in the presence of probabilistic rebroadcast at the intermediate motes. This is because, when there is a large number of new packets being inserted at the source, there is an increase in the number of retransmissions at the other nodes, and this leads to collision due to hidden terminal problems or other interference issues.

The effect of interference is also observed in our experimental results shown in Figure 8, where we compare the reliability of FBcast with $n = 20$ for 121 motes deployed at 10 feet inter-mote spacing. When the data source injects roughly one packet per second, we observe that reliability suffers heavily at high forwarding probability: though the number of retransmissions shown in Figure 8-B is very high, the number of successful receipt is low (see Figure 8-A). However, when the data source slows down the rate of packet broadcast (a packet roughly every 2 seconds), the reliability increases continuously until it reaches 100% at higher forwarding probabilities. The amount of transmission overhead increases, but so does reliability, indicating that the interference effect is subdued because of the slower rate of source data broadcast. The effect of interference is less apparent for higher stretch factors because of the basic property of FEC-based data recovery, i.e., even if some of the packets are lost due to interference, other motes will be able to reconstruct the original data.

Need for FBcast Extension. From the above results, it is clear that FBcast can be adapted more flexibly to suit different deployment densities and reliability requirements. However because of the number of parameters involved, the complexity of packet loss characteristics, and the probabilistic nature of FBcast, there is no simple expression that captures FBcast reliability for different parametric settings and network conditions. In the following discussion we explore how we can achieve high reliability for various network sizes without dynamically adapting the stretch factor or forwarding

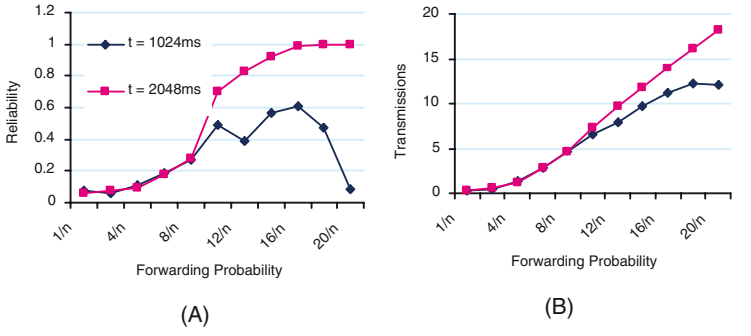


Fig. 8. The effect of injection rate: how interference causes the reliability to drop at higher forwarding probability, though the amount of retransmissions increases as expected

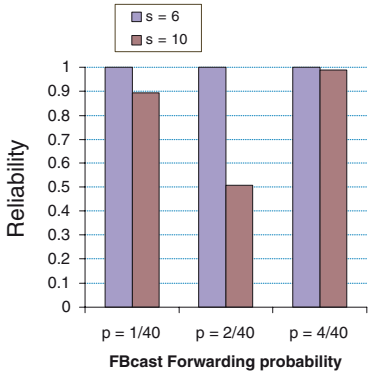


Fig. 9. Performance of FBcast ($n = 40, p = 2/40$) with repeaters for 441 motes deployed with inter-mote density s = 6' and 10'

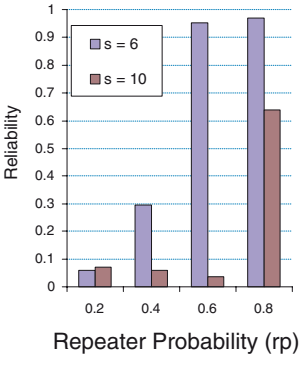


Fig. 10. Performance of pbcast ($p = 0.8$) with repeaters for 441 motes deployed with inter-mote density s = 6' and 10'

probability. In doing so we look at the limitation of FBcast in covering large deployments, which leads to our solution using repeater extensions.

4.4 Protocol Extension with Repeaters

In the experiments in the Section 4.3, all the motes are within the broadcast range of the source (referred to as single-hop experiments). The network topology used here is once again a grid of motes, but unlike the earlier single hop experiments, here the mote density is kept the same while increasing the number of motes, thus expanding the deployment area. For example, a grid of 441 motes deployed with inter-mote spacing of s = 10', will cover 200' X 200' area. With increase in the deployment area, the number of hops between the data source and the peripheral motes increases, realizing the effect

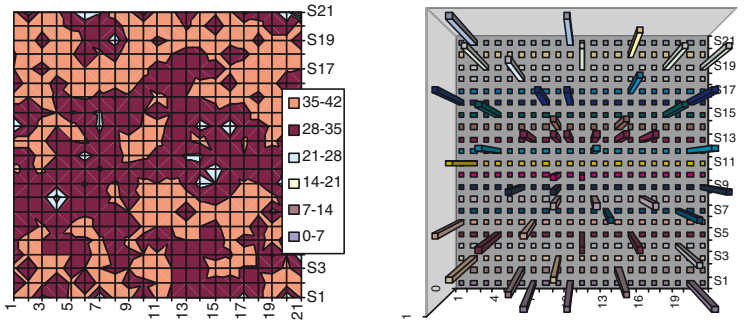
of multi-hop communication. In the presence of such multi-hop communication, we want to measure the reliability of *pbcast* and FBcast protocols. For the following experiments, $pbcast$ is set with $p = 0.8$ because $pbcast$ with lower forwarding probability value has very low reliability. Experiments reveal that even though, FBcast provides higher reliability than $pbcast$, the reliability decreases with increasing deployment area. The fraction of motes being able to reconstruct the original data decreases with increase in the deployment area. There are two possible reasons for this result. First, hidden terminal problem is more severe here than in the single-hop experiments. For example, for a small deployment area, the source mote was found to be able to inject all 10 packets, but for a larger deployment area, the source mote had to retry injecting the original packets several times. Second, the peripheral motes are able to receive only a few or no packets.

Because of channel loss and probabilistic retransmission, as we go away from the data source in the center, the number of received packets decreases. This is observed in single hop scenario also (see Figure 3), but it is more evident for multi-hop scenario. For these experiments, the inter-mote spacing is 10'. With 441 motes placed uniformly in a 200'x200' area, the figure shows the number of packets received in different zones. For *pbcast*, with $p = 8/10$, the broadcast coverage is less than 5% of the area. As we increase n, we observe an increase in the coverage. But increasing n also has inherent cost (encoding/decoding cost), a very high n may not be the desirable engineering choice. Also, even with $n = 60$, the coverage is less than even 20%. Next, we explore how extending FBcast with repeaters extends the broadcast coverage.

A repeater is a mote that reconstructs the original data and acts as the data source, thus injecting the encoded data packets. For $pbcast$, being a repeater just means retransmitting the original data packets, and for FBcast, being a repeater means decoding the received packets to reconstruct the original data, encoding the data again to generate n packets, and re-injecting all the packets. Hence, only a mote that has received at least k packets can be a repeater. We design and evaluate an FBcast protocol with repeater motes. For a fair comparison of $pbcast$ and FBcast, we also develop a repeater variant of $pbcast$ and compare it with FBcast.

FBcast Extension with Repeaters. Because of unknown data source mote, unknown network topology and radio behavior, and probabilistic nature of the broadcast, a priori placement of repeaters is not desirable. A repeater should be selected dynamically, and such a selection poses a question: how can a mote decide to be a repeater based on the received packets? If the condition for being a repeater is very relaxed, there may be too many motes serving as repeaters (over-coverage), and if the condition is very tight, there will be too few repeaters to cover the whole area (under-coverage). Our repeater algorithm strikes a balance by utilizing the rate of packet receptions and the number of received packets.

The repeater algorithm works as follows. Every mote calculates how long it should listen before it decides about becoming a repeater, call this time is the *listen window*. At the end of a listen window, if a mote has enough packets to construct original data, but less than a threshold number of packets (k_{th}), the mote becomes a repeater. By having threshold check as a condition, we ensure that not every mote becomes a repeater, but only those that are able to receive a small fraction of the injected packets. This threshold



A. Topography for multi-hop scenario with repeaters (200'x200' area with grids of size 10'10')

B. Position of the repeater motes

Fig. 11. FBcast with repeaters for motes deployed with 10' spacing in a 200'x200' area: (A) shows the coverage, and (B) shows the overhead in terms of number of repeaters and their positions

condition will be satisfied by a band of motes, as it is clear from the one-hop results; around the data source, there exist concentric bands of motes with similar number of received packets. To ensure that not all motes in a particular band become repeates, we carefully randomized the length of initial listen window. Listen window consists of two parts: first part is the expected time when a mote will receive the threshold number (k_{th}) of packets, and the second is a random duration from 0 to $t \in [0, t_k]$, where t_k is the duration between reception of first and kth packets. The length of listen window affects the latency of packet dissemination in large area. For faster propagation, the listen window can be reduced; though the flip side of this choice is an increase in the number of repeaters per unit area.

Figure 9 shows the results of FBcast deployed over a coverage area of 200'x200' with motes spaced every 10' along the grid points. Reliability is plotted on the Y-axis against the forwarding probability on the X-axis. We present the results for 2 different spacings, $s = 6'$ and $s = 10'$. We see that for most of the cases the attained reliability is very high, except for one instance (because of the probabilistic nature of the protocol). The algorithm parameters are set as follows: $n = 40$, $p = 8/40$, and $k_{th} = 21$. The value of threshold packet count, k_{th} is important. Since a mote becomes repeater only if it receives packets between k and k_{th}, setting k_{th} too close to k decreases the probability of a mote becoming a repeater.

In Figure 11A we show the topographical coverage attained by FBcast with repeater; for the chosen setting, complete coverage of the 200'x200' area is attained. Most of the motes receive more than 21 packets. We found that setting $k_{th} = 1.5k$ for mote deployments with 10' spacings or less enables this complete coverage. Figure 11B shows the position of repeater motes.

pbcast Extension with Repeaters. For $pbcast$, typically there is no way for a mote to know how far it is located from the data source (unless some extra information such as location about the neighborhood is provided). Thus, in this variant we assign a predefined probability (rp) of being a repeater to any mote that has received 10 packets.

Figure 10 shows that $pbcast$ with repeaters can provide complete coverage of the deployment area, albeit at the cost of high repeater probability. For $rp = 0.6$, the reliability is 85% for inter-mote spacing of $s = 6'$, but the reliability goes below 10% for spacing of $s = 10'$ (sparser mote density). Increasing the repeater probability helps achieving high reliability for sparser deployments, but that also increases interference; high values of rp essentially amounts to data flooding, and the consequent interference leads to the well known broadcast storm situation [14]. At an inter-mote spacing of 10', we noticed more than 99% coverage only for very dense repeater deployments ($rp > 0.8$).

Protocol Comparisons. The repeater variants of $pbcast$ and FBcast both have higher reliability compared to their original counterparts. For sparser deployment, $pbcast$ yields high reliability only with very high repeater probabilities, thus causing high transmission overhead. FBcast (with the aid of listen window and threshold number of received packets) is able to control the number of repeaters while ensuring more than 99% reliability for various deployment parameters.

5 Conclusion

We have presented a new broadcast protocol that exploits data encoding technique to achieve higher reliability and data confidentiality at low overhead. The simulation experiments show that with increased network density, traditional broadcast become quite unreliable, but FBcast maintains its reliability. The forwarding probability parameter of FBcast can be tuned to decrease the number of transmissions with higher density. FBcast with repeaters allow nodes to reconstruct the original data and then re-inject the new packets into the network (when the number of received packet falls below a threshold). FBcast trades off computation for communication. The data encoding (source) and decoding (recipients) consume computation cycles, but since computation is order of magnitude less power-expensive than communication, we expect to save power. Also, considering the computation needs of the encoding scheme, FBcast is suitable for computationally rich nodes. Based upon the continuing trend we believe that today's handhelds are tomorrow's motes, and FBcast will be quite suitable for future WSN.

References

1. P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.
2. D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks, 2002. Technical Report, Intel Research.
3. Jae-Hwan Chang and Leandros Tassiulas. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the First ACM/Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
4. P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 126–137. ACM Press, 2003.

5. L. Li, J. Halpern, and Z. Haas. Gossip-based ad hoc routing. In *Proceedings of the 21st Conference of the IEEE Communications Society (INFOCOM'02),.*, 2002.
6. H. Lim and C. Kim. Flooding in wireless networks. *Computer Communicatins*, 24(3-4):353–363, 2001.
7. W. Lou and J. Wu. On reducing broadcast redundancy in ad hoc wireless networks. *IEEE Transactions on Mobile Computing', volume =.*
8. M. Luby. Lt codes. In *Proceedings of 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.
9. P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.
10. K. Obraczka, K. Viswanath, and G. Tsudik. Flooding for reliable multicast in multi-hop ad hoc networks. *Wireless Networks*, 7(6):627–634, 2001.
11. L. Orecchia, A. Panconesi, C. Petrioli, and A. Vitaletti. Localized techniques for broadcasting in wireless sensor networks. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 41–51. ACM Press, 2004.
12. S.-J. Park, R. Vedantham, R. Sivakumar, and I. F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 78–89, New York, NY, USA, 2004. ACM Press.
13. W. Peng and X.-C. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 129–130. IEEE Press, 2000.
14. Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. *Wirel. Netw.*, 8(2/3):153–167, 2002.
15. C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: a reliable transport protocol for wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 1–11, New York, NY, USA, 2002. ACM Press.
16. B. Williams and T. Camp. Comparison of broadcasting techniques for mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 194–205. ACM Press, 2002.