# FOUNTAIN BROADCAST FOR WIRELESS NETWORKS

*Rajnish Kumar* *Arnab Paul* *Umakishore Ramachandran*

College of Computing
Georgia Institute of Technology, Atlanta
rajnish, arnab, rama@cc.gatech.edu

## ABSTRACT

*Code dissemination in wireless ad hoc network is an important aspect of network deployment. Once deployed, the network nodes may still need software updates to keep up with the changing application demand, thus making wireless broadcast an important aspect of any wireless network deployment. We present FBcast, a new broadcast protocol based on the principles of modern erasure codes. FBcast provides high reliability and data confidentiality: often considered critical for disseminating codes. 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 ad hoc sensor network (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.

Deploying an efficient wireless broadcast would face two key interrelated challenges:

(*i*) *Messaging Overhead :* Traditionally, every node rebroadcasts any new data packet, resulting in too many unnecessary transmissions [16]. For example, if a software update of $k$ packets is to be sent over a WSN of $n$ nodes, potentially $k.n$ broadcasts could be sent out. Larger the number of broadcasts, more is the cumulative power consumed because of the communication. Furthermore, the increased messaging overhead introduce more collisions and thus affect the channel reliability.

(*ii*) *Reliability :* Reliability of message dissemination is a key requirement for a sensor network to function properly. 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 show the lossy nature of wireless channels [3], and achieving high reliability in presence of channel loss and collisions becomes a challenge.

**Table 1**. Hardware Platform Evolution

| Mote | WeC | dot | mica2 | iMote |
|---|---|---|---|---|
| Released | 1999 | 2001 | 2003 | 2003 |
| Processor | 4 MHHz | 4 MHz | 7 MHz | 12 MHz |
| Flash (code, kB) | 8 | 16 | 128 | 512 |
| RAM (kB) | 0.5 | 1 | 4 | 64 |
| $\mu$controller | Atmel | Atmel | Atmel | ARM |

So far, two baseline approaches have been proposed in the literature - deterministic and probabilistic flooding. It turns out that simple deterministic flooding protocols are quite inefficient to address the issues mentioned above. The probabilistic approach improves upon it; each node randomly decides whether or not to broadcast a newly seen data packet. These baseline approaches do not assume any extra information on the networks. Several variants and optimizations over these two baseline schemes have also been introduced. Typically, these derivatives either assume some structural information about the networks, *e.g.,* knowledge of the network neighborhood, inter-node 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 [16, 15, 18, 7].

However, it may not often be possible to depend upon any additional information for reasons that are very typical 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 and location *etc.,* may continue to change due to mobility and failures. The periodic Hello gossip becomes expensive to support because of the transmission overhead incurred and dynamic nature of the WSN.

We propose a new baseline protocol that is based on a fundamentally different principle, that of the erasure codes. [1] Like the other baseline protocols, the new protocol does not assume any structural knowledge about the sensor networks, and needs no extra round of messaging such as ACK packets. We show

[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.

that the protocol achieves higher reliability compared to the basic probabilistic broadcast, but at a much lower messaging cost.

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. Table 1 presents a quick summary of the currently available sensor platforms [6]; 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 posses 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 benefit in terms of smaller messaging overhead, less collision, enhanced reliability and reduced power consumption.

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. Founded on this principle, FBcast, a new WSN protocol, offers high reliability and low messaging overhead.

(*ii*) We compare FBcast with probabilistic broadcast through simulation studies using the TOSSIM simulator [5]. Protocol efficiency can be evaluated over more than one axes, 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 fare comparison between these approaches may not always be possible. However, our experiments do suggest that, by and large, 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 and verify the correctness via simulation.

The paper is organized in the following way. Section 2 provides details of FBcast protocol. It also explains the encoding scheme used. Section 3 presents the implementation details of FBcast and preliminary results. Section 4 looks at the related broadcast protocols to place our work. At the end, we present future work and conclude.

## 2. FBCAST PROTOCOL

Figure 1 and 2 represent the overview of FBcast broadcast protocol. The data to be disseminated consists of $m$ packets. The source blows up these $m$ packets into $n$ packets, using 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 ( $k \geq m$ ) as required for decoding, it reconstructs the data and passes it to the application. It is assumed that all network nodes share a seed value information used for the data encoding and decoding.

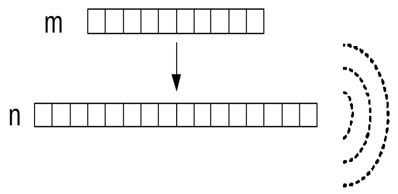


**Fig. 1**. FBcast at source: $m$ original packets are encoded into $n$ packets and injected in the network.

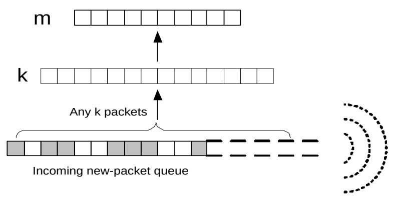


**Fig. 2**. FBcast at recipients: $k$ packets are chosen from received queue and decoded back to $m$.

### Encoding schemes

Erasure codes provide the type of encoding that is needed to accomplish the protocol. However, some of the desirable properties set apart *Fountain Codes* (a modern class of erasure codes [1]) from the rest: (*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 needed, by a decision that can be taken online. (*ii*) There is no restriction on the packet length. (*iii*) Fountain codes are quite inexpensive in encoding (linear) and decoding times ($n \log n$). Standard erasure codes, such as the Reed-Solomon codes are very *inflexible* in the first two aspects, and quite inefficient in performance.

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. 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 $\epsilon$, only $k = (1 + \epsilon)m$ number of data blocks out of this fountain suffice to reconstruct the original document. An example of a rate-less code is the Luby-Transform codes [9]. Our idea is to generate the blocks and sprinkle them over to the neighbors, which in turn need to re-sprinkle only 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 the 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 wireless channel, code confidentiality is often a requirement of 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.

## 3. 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 on Linux to tune the FBcast parameter (e.g. stretch factor). To understand the protocol behavior, we simulated FBcast using TOSSIM [5] for different network sizes. We used the empirical radio model supported in TOSSIM, thus every link is used as a lossy channel with loss probability based upon empirical data. 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 [3]. The lossy radio model generated using $LossyBuilder$, instead of using a perfect model, allows us to see the effect of packet loss at the broadcast. 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 all of the injected packets. If a mote receive 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 total number of transmission in the network 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 $p = 0.8$.

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 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 [9]). $\delta$ is a small constant (*e.g.*, $\delta = 10^{-3}$ gave 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 the 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.

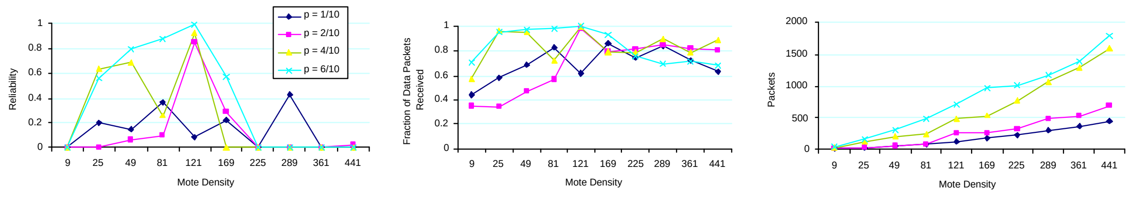
The rest of this section is organized as follows. First we explore the broadcast protocol behavior for one-hop networks. Our results suggest that even if all nodes are situated within the transmission range of the source, the number of received packets fall down with distance from the source. Thus, we need repeaters for larger networks, which we then explore in the context of multi-hop networks.

### 3.1. One-hop Network

The network topology used is a grid of motes put in an area of 50 feet by 50 feet, with the data source mote at the center. As the number of motes increases, the distance between the motes decreases, *i.e.* for a grid of $w^2$ motes, the distance between two adjacent motes is set to $50/(w - 1)$ feet. The transmission range of used radio model is 50 feet. Thus, every node is within transmission range of the data source, but because of channel loss and collisions, motes may not receive all data packets in single transmission. The following experiments explore how broadcast protocols behave with change in mote density. Simulation was allowed to run for enough time until there was no more broadcast transmission going on in the network. Specifically, the simulation was run for 10 seconds after the network became idle (no transmission). For most of the experiments with one-hop network, this time was about a minute. Below, we first look at the *pbcast* results, then FBcast, and finally their comparisons.

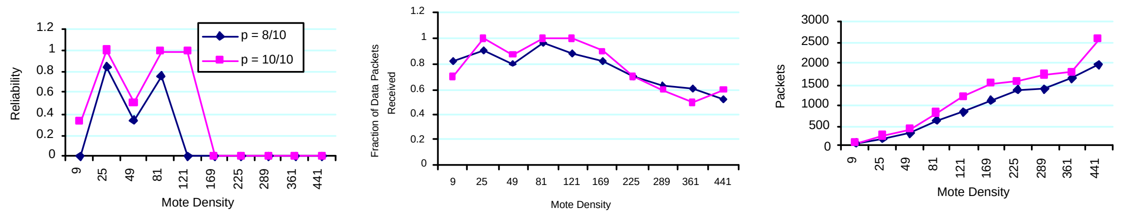
#### *3.1.1. Probabilistic Broadcast Results*

Here, we present $pbcast$ performance for varying mote density in the same area. Each point in the results correspond to a single TOSSIM run, *i.e.* they are not averaged over multi-



(A) Reliability Vs Mote Density (B) Packets Received Vs Mote Density (C) Number of Transmissions Vs Mote Density

**Fig. 3**. Probabilistic broadcast results for low forwarding probability $(p = 0.1 - 0.6)$. Motes are uniformly deployed as a grid over 50'x50' area. 10 packets are injected at the source at center.



(A) Reliability Vs Mote Density (B) Packets Received Vs Mote Density (C) Number of Transmissions Vs Mote Density

**Fig. 4**. Probabilistic broadcast results for high forwarding probability $(p = 0.8 - 1.0)$.

ple TOSSIM runs with same parameters. Presenting results for a typical TOSSIM run shows some randomness in the results, which could be removed by taking averages. But, since we want to focus upon the broadcast reliability, a typical run instead of an average is a more desirable study.

Figure 3A and 4A show that reliability of traditional probabilistic broadcast is more than 90% only when $p$ is more than $6/10$, and it holds only for a small window of mote density variation *i.e.,* when there are about 200 motes in 50'x50' area. Reliability here presents the fraction of motes that are able to reconstruct the original data, *i.e.* they receive all of the 10 injected packets. When network is sparse, reliability is low because of channel loss. When network is dense, again reliability is low because of packet collision. Figure 3A and 4A also show that increasing forwarding probability does not always mean an increase of the reliability. This can be explained by the fact that increased forwarding probability leads to increased transmission attempts by the motes, thus increasing the probability of packet collision. To understand the reliability behavior closely, we looked at the number of new packets being received by the motes. Figure 3B and 4B show the effect of mote density and forwarding probability upon the number of packets received. The number of received packets vary similar to the reliability in Figure 3A because they are related.

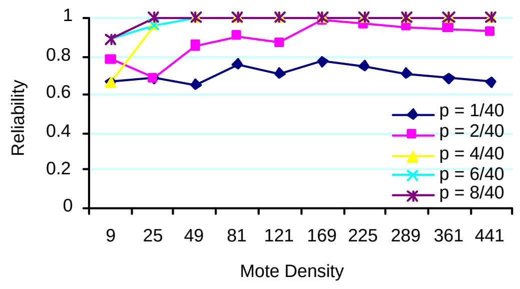
Transmission overhead of *pbcast* protocol increases with increase in forwarding probability and with increase in mote density as shown in Figure 3C and 4C. The average latency, when a mote receives all 10 packets if at all, remains same for different forwarding probabilities. This is mainly because, in our experiments, the latency is found to be bounded by the rate at which the data source injects the packets. We will use the reliability and transmission overhead results for comparing *pbcast* and FBcast performance later on.

### *3.1.2. FBcast Results*

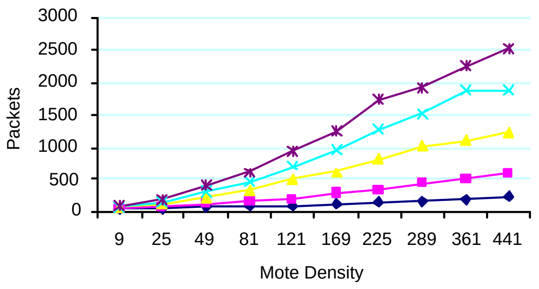
Figure 5A shows that FBcast reliability, the fraction of motes that are able to reconstruct original data, is close to 100% for a large window of mote density variation: when mote density is more than 49, and $p$ is more than $4/40$, all motes are able to receive more than 14 packets. These results are for $n = 40$. FBcast has high reliability compared to *pbcast* because motes are able to reconstruct the original data even if they do not receive all $n$ injected packets. Below we look closely at the effect of stretch factor and forwarding probability upon FBcast behavior.

We experimented with the effect of varying the forwarding probability, $p$, upon reliability (Figure 5A). As we decrease the $p$ value, the number of retransmissions decreases as shown in 5B, but the number of new packets received at different motes decreases. For dense network, smaller value of $p$ is sufficient to reconstruct the original data; *e.g.,* for 169 mote network in 50 x 50 feet area, $p = 2/40$ is enough to have more than required number of new packets for 99% motes. But for less dense network, higher value of $p$ is required. Thus, if the deployed mote density has very high variance, the value of $p$ should depend upon the neighbor density. Second, as we go away from the data source, the number of new packets received decreases. This suggests that after some hops from the source, there may not be enough new packets to reconstruct the original data. Increasing the stretch factor helps overcoming this shortage, as shown later in this section.

Number of transmissions, shown in Figure 5B, includes the number of injected packets and number of retransmissions un-

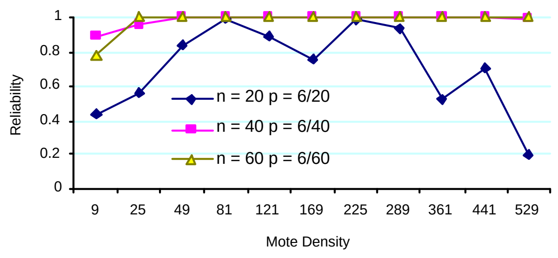


(A) Reliability Vs Mote Density

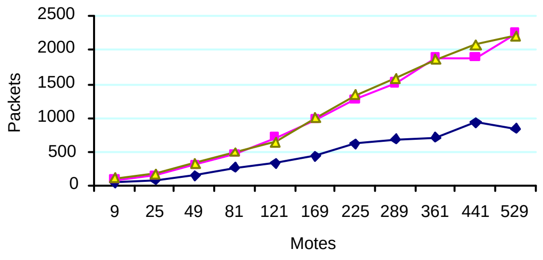


(B) Number of Transmissions Vs Mote Density

**Fig. 5**. FBcast performance for $n = 40$ and varying forwarding probability. Motes are uniformly deployed as a grid over 50'x50' area.



(A) Reliability Vs Mote Density



(B) Number of Transmissions Vs Mote Density

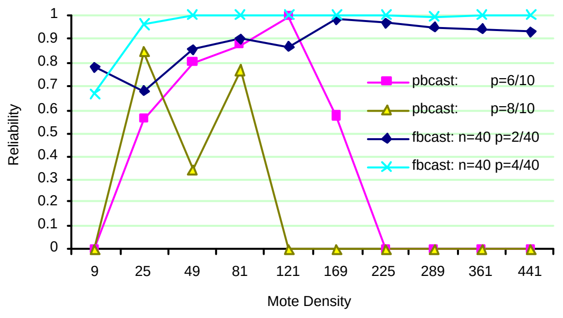
**Fig. 6**. FBcast performance for $p = 6/n$ and varying stretch factor.

til the simulation run ends. For the same forwarding probability, there is an increase in transmission overhead when stretch factor is increased from n=20 to n=40, but there is no change between n=40 and n=60. Though the forwarding probability is proportionately adjusted, change in transmission overhead with $n$ can be explained by the corresponding change in reliability shown earlier: reliability improved from n=20 to n=40, but it remained roughly the same for n=40 and n=60. But for the same stretch factor, n=40, if we increase the forwarding probability, we observe an increase in the number of transmissions, though there was no similar increase in the reliability. To understand this, we looked at the total number of packets being received by the motes (not the reliability), and we found that there is an increase in the number of received packets at the motes with an increase in the forwarding probability. For example, for $p = 1/40$, the average number of received packets for 121 motes is 23, and for $p = 8/40$, the average increases to 38, though in both cases most of the motes have enough new packets to reconstruct the original data.

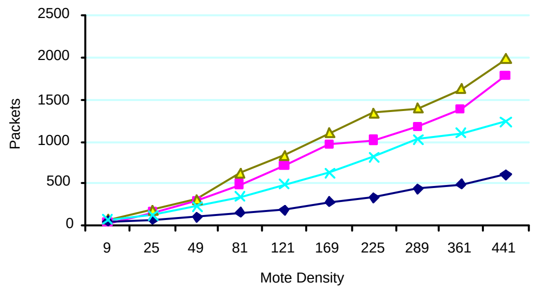
To see the effect of stretch factor, $n$ is varied from 20 to 60, and the forwarding probability ($p$) is varied proportionately with the stretch factor. This variation ensures that a high stretch factor does not introduce too many unnecessary packet retransmissions. However, a high stretch factor injects many different packets and thus the intermediate nodes will continue to get more *new* packets to retransmit. The combined effect of adjusted $p$ and stretch factor ensures that higher reliability is achieved with approximately similar amount of transmission. The results in Figure 6A show that for a sparse network, increasing $n$ improves reliability, but for a dense network, we see that $n = 40$ provides very high reliability and there is no need for higher stretch factor, of course there is an increase in number of transmissions with increase in the stretch factor (Figure 6B).

Latency numbers show an increase in the average latency as we increase the stretch factor. This can be explained because the encoded packets are injected in sequence, and the higher the number of injected packets, the longer will it take to inject the last packet. For same stretch factor, there is not much difference in average latency for different forwarding probability.

In summary, FBcast has very high reliability for $n = 40$, and there is no need for higher stretch factor for fairly dense network (100 motes in the 50'x50' area). Also, for higher density, $p = 4/40$ achieves high reliability but with lower number of transmissions than $p = 8/40$. The average communication latency in FBcast depends upon how fast the data packets are being injected in the network.



(A) Reliability Vs Mote Density



(B) Number of Transmissions Vs Mote Density

**Fig. 7**. Comparison of FBcast and $pbcast$ for reliability and transmission cost.

### 3.1.3. *Protocol Comparisons*

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 the *pbcast* reliability is highly susceptible to the mote density. Since both FBcast and *pbcast* use proportional forwarding probability to control the number of retransmissions, both have similar transmission overhead for same forwarding probability; but since FBcast provides high reliability at lower forwarding probability than $pbcast$ (Figure 7), the number of retransmissions incurred in FBcast is much less than that in $pbcast$. 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.

## 3.2. Multi-hop Network

Because of channel loss and probabilistic retransmission of the arrived packets, as we go away from the data source in the center, the number of received packets decreases. This is observed in single hop scenario too, but it is more evident for multi-hop scenario as shown in Figure 8. For these experiments, the inter-mote spacing along grid sides is 10'. With 441 motes placed uniformly in 200'x200' area, the figure shows the number of packets received in different zones. The *pbcast*, with $p = 8/10$, the broadcast coverage is less than 5% of the area. As we increase $n$, we observe 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%. Below we present how extending FBcast with repeaters extends the FBcast coverage.

A repeater is a mote that reconstructs the original data and acts as the data source, thus injecting the encoded data packets. 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 more motes serving as repeaters than is required (over-coverage), and if the condition is very constrained, there may 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.

Figure 9 shows the pseudo code for FBcast with repeater algorithm. Every mote calculates how long it should listen before it decides about being a repeater, call this time is the listen window. At the end of listen window, if a mote has enough packets to construct original data, but less than a threshold

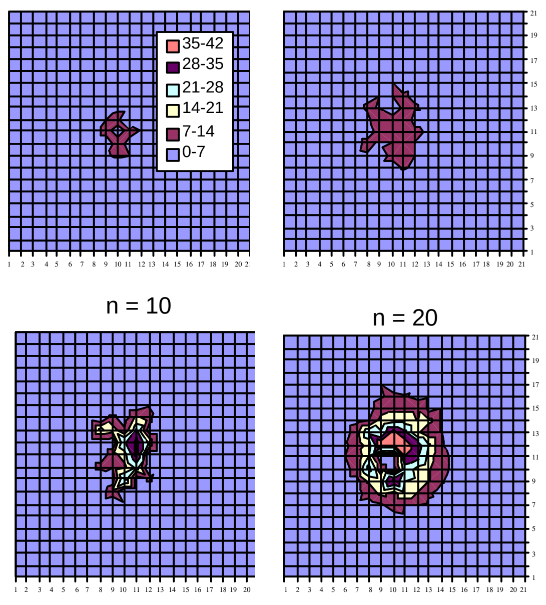
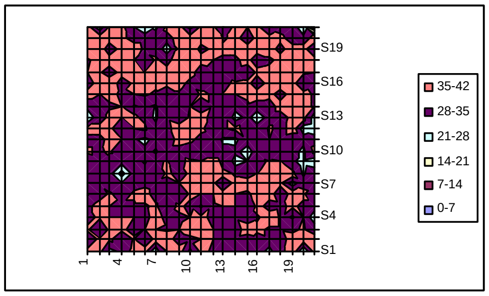
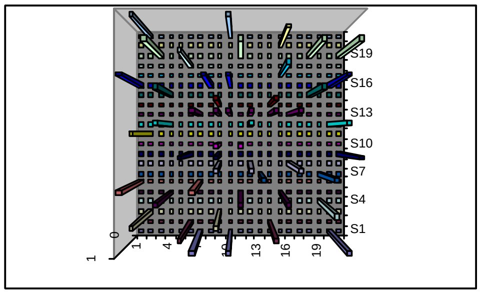


**Fig. 8**. Topography for motes deployed with 10' spacings in 200'X200' area. $n$ varies from 10 *pbcast* to 60. Transmission probability varies as $8/n$.



A. Topography for multi-hop scenario with repeaters (210'x210' area with grids of size 10'10')



B. Position of the repeater motes

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

```
Input: k, $k_{th}$, packet arrival times
Algorithm:
$t_1$ = time when 1st packet received
$t_k$ = time when $k$th packet received
At $t_k$:
    • compute $t_{k_{th}} = (t_k - t_1) * k_{th}/(k-1)$
    • compute $t_{rand}$ = random value in $\{0, t_k\}$
    • compute $listenWindow = t_{k_{th}} + t_{rand}$
Wait for $listenWindow$ duration;
If( total packets received $> k_{th}$ )
return; // no need to be a repeater
Else if( total packets received $>$ k ) // be a repeater
    1. reconstruct original data
    2. encode data to get all $n$ packets
    3. transmit the new packets
```

**Fig. 9**. Pseudocode for FBcast with repeater.

number of packets, the mote becomes a repeater. By having threshold check as a condition, we ensure that not every mote can become a repeater, but only those that are ( being away from the data source ) able to receive only a small fraction of injected packets will be repeaters. This threshold condition will be satisfied by a band of motes, as it is clear for one-hop scenario: around the data source, there exists concentric bands of motes with similar number of received packets. To ensure that not all motes in the distant band become a repeater, we carefully randomized the length of initial listen window.

Listen window consists of two parts: first part is the expected time when the 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 $k$th packets. The length of listen window affects the latency of packet dissemination in large area. For faster propagation, the listen window can be proportionately decreased to smaller value, though the trade-off is the increase in number of repeaters per unit area.

Figure 10A shows that FBcast with repeater has complete coverage of the 200’x200’ area with motes kept at 10’ spacing. 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 to be mentioned. Since a mote becomes repeater only if it receives packets between $k$ and $k_{th}$, setting $k_{th}$ too close to $k$ value decreases the probability of motes being a repeater. We found that setting $k_{th} = 1.5k$ for mote deployments with 10’ spacings or less enables complete coverage. Figure 10B shows the position of repeater motes.

## 4. 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 [11] is the most naive implementation of the first class. However, naively rebroadcasting can easily lead to broadcast storm problem [16], and hence the need for controlled density-aware flooding and multicast algorithms for wireless networks [2].

**Variants and Optimizations:**Several other optimizations can be applied to these baseline schemes. For example, Location awareness can be exploited to subvert issues such as the hidden neighbor problem. However, such a scheme would require more sophistication such as communication with a satellite for GPS positioning. For a mobile environment, a location aware protocol can be quite useful [16]. Similarly, knowing one’s neighbors within a first few hops can help broadcasting efficiently as well. In this case, neighbors exchange periodic Hello messages amongst each other so as to form an idea of their neighborhood. When a broadcasting is done, the message is tagged with the identifier of the broadcasting node so that neighbors within a threshold number of hops need not rebroadcast them. Examples of such techniques can be found in the literature [8, 13]. 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 [12]. For a comparative and comprehensive survey of these protocols, the reader can refer to [18]. A randomized variant of controlled flooding is by the epidemic and gossiping algorithms for data management in large-scale distributed systems [14]. On a similar note, Trickle

[4] combines epidemic algorithms and density-aware broadcast towards code dissemination in WSN. Another variant of gossiping over *pbcast* is the Fireworks protocol [12].

**FBcast and Erasure Codes:** 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. FBcast is inspired by the already known use of erasure codes in the domain of digital storage and transmissions. It is a well understood concept that compared to naively replicating data, an erasure code can provide similar reliability at a much smaller space-overhead [17]. Several distributed and peer-to-peer storage systems have been built on this principle. In wireless broadcasting, the two existing baseline approaches (simple broadcast and *pbcast*) are in a sense similar to *naive replication*, *i.e.* during a broadcast a data packet is fully replicated. An erasure code however, breaks a packet into smaller blocks and introduces redundancy in a tricky way that is lot more space-efficient. Thus using FBcast ensures that reliability comes in-built within the original data packets, and therefore less support is needed from the messaging subsystem.

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 [1]. Later on, versions of such codes have been shown as useful tool for Peer to Peer file sharing/download purposes [10]. FBcast applies the idea of fountain encoding with the previously known scheme of probabilistic gossip, to achieve high reliability without extra messaging overhead, in the domain of wireless networking where bandwidth, power and reliability are very critical issues to be addressed.

## 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.

Currently FBcast assumes same forwarding probability for whole network. We are looking into having a variable forwarding probability for different motes such that it depends upon the local network density. Also, we observed that as we move away from the data source, the number of received new packets decreases. We have proposed a repeater algorithm that reconstructs the original data and then re-injects 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.

## 6. REFERENCES

[1] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM*, pages 56–67, 1998.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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 ACM/Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[7] 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.

[8] H. Lim and C. Kim. Multicast tree construction and fboding in wireless ad hoc networks. In *Proc. ACM Workshop on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, 2000.

[9] M. Luby. Lt codes. In *Proceedings of 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[10] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[11] K. Obraczka, K. Viswanath, and G. Tsudik. Flooding for reliable multicast in multi-hop ad hoc networks. *Wireless Networks*, 7(6):627–634, 2001.

[12] 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.

[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] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[15] I. Stojmenovic and J. Wu. Broadcasting and activity -scheduling in ad hoc networks, 2004.

[16] 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.

[17] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Peer-to-Peer Systems: First International Workshop (IPTPS)*, 2002.

[18] 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.