ACR: Active Collision Recovery in Dense Wireless Sensor Networks

Yafeng Wu, Gang Zhou†,John A. Stankovic
Computer Science Department, University of Virginia, {*yw5s,stankovic*}*@cs.virginia.edu*
†Computer Science Department, College of William and Mary, *gzhou@cs.wm.edu*

***Abstract*—Packet collision causes packet loss and wastes resources in wireless networks. It becomes even worse in dense WSNs, due to burst-traffic and congestion around sinks. In this paper, we propose a novel protocol to recover collided packets. Our experiments on a testbed reveal that collisions between long packets and short packets cause a partial error pattern on collided packets, which can be used for efficient recovery. We give a theoretical analysis that demonstrates that combining such collision recovery with CSMA protocols achieves a significant performance improvement. Then, we design ACR, an Active Collision Recovery protocol, which actively converts most potential collisions into LS-collisions, and then applies a lightweight FEC scheme to recover collided packets with such partial error patterns. We implement ACR on a Tmote testbed, and compare its performance with other packet recovery schemes. Results show that ACR significantly reduces the number of retransmissions, and achieves around 25% improvement on transmission efficiency over other schemes.**

I. Introduction

Packet Collision occurs when two or more close stations attempt to transmit a packet at the same time. This can result in packet loss and impede network performance. Many CSMA based MAC protocols are proposed in Wireless Sensor Network (WSNs) to avoid collisions, such as B-MAC [1] and X-MAC [2]. These protocols can efficiently reduce collisions, but intrinsically cannot eliminate all collisions, because of hidden terminal problems, as well as collisions when multiple nodes sense the medium free at the same time. Previous work [3] demonstrates that CSMA based protocols cannot avoid such collisions when the number of concurrent transmissions grows. Such collisions become severe in dense WSNs for two reasons. First, many dense sensor networks are event-driven and generate bursty spatially-correlated traffic, where multiple sensors in the same neighborhood all have messages to send simultaneously in response to the same event. Second, WSNs are typically equipped with few sinks to which packets from sensors converge. Such convergence cause many collisions around sinks, described as the "funneling effect" [4]. Furthermore, the consequences of packet collisions are serious to WSNs. Collisions can cause the loss of critical control information from base stations, and applications may fail.

When collisions still occur, a packet recovery method should be applied to recover lost packets. The traditional Automatic Repeat Request (ARQ) method uses acknowledgment and retransmissions for packet recovery. However, ARQ brings high latency due to waiting for ACKs, and retransmissions cause more collisions, especially with heavy traffic. New packet recovery methods exploit the Partial Corrupted Pattern (PCP), where only some bits are actually corrupted in a "lost" packet. For instance, Forward Error Correction (FEC) helps receivers recover such partially corrupted packets by sending packets with redundant bits. As shown in [5], it significantly reduces retransmissions and delivery latency. However, since these protocols are only used to recover corrupted packets caused by lossy links, it is still unclear if such methods can be used to recover packets corrupted by collisions in WSNs.

In this paper, we focus on how to efficiently recover collided packets in dense WSNs, with bursty and heavy collisions. First, we investigate PCP due to collisions through experiments on a testbed of Tmote sensors. Unfortunately, our experiment results show that almost all bits of collided packets are erroneous in typical WSNs where nodes use CSMA and send packets with similar sizes. These results indicate that current packet recovery methods cannot deal with the collision recovery. However, when nodes send packets with two different sizes, we observe that only certain bits in the long packet are corrupted in collisions between long and short packets, and this chunk of erroneous bits has almost the same size of the short packet. We call such collisions LS-collisions. LS-collisions produce a regular PCP in collided packets, which can be exploited to achieve efficient collision recovery. Therefore, such collisions are quite "useful". Our key insight is that if collisions cannot be totally eliminated under CSMA, it is better to have more LS-collisions, because we can recover collided packets from them.

In order to justify this idea, we provide a thorough analysis based on the model derived from scenarios where N senders compete for transmission using CSMA. The analysis demonstrates that by exploiting LS-collisions, we can achieve a higher *probability of successful transmission*, as well as a much higher *transmission efficiency* defined as the ratio of successfully transmitted information bits to overall bits transmitted. The analysis also gives insights on how to choose the right proportion of nodes sending long packets and short packets, and a sub-optimal probability distribution for senders to randomly select contention slots that maximizes the above two metrics.

We then presents ACR, an *Active Collision Recovery* protocol that mitigates the negative impact of collisions by

efficiently recovering collided packets. Unlike other existing recovery schemes, ACR is "active" in that it actively transforms potential collisions into LS-collisions to maximize the recovery probability, rather than passively waiting for collided packets to be recovered. ACR then uses a block based FEC scheme to efficiently recover corrupted packets due to LS-collisions. To identify erroneous blocks, ACR employs a novel RSSI-based error detection method, which does not introduce extra checksum overhead. In case of recovery failures, ACR also has a backup ARQ scheme. Unlike ZigZag[6] or PPR[7], ACR does not require customized hardware. ACR can be easily integrated into existing CSMA protocols with off-the-shelf sensor devices. Our main contributions are:

- An empirical study on bit error patterns in collided packets and the revelation of LS-collisions, which enables achieving efficient collision recovery.
- A theoretical analysis which demonstrates that we can achieve a significant performance improvement by exploiting LS-collisions under CSMA.
- A design of a novel ACR protocol, which actively creates LS-collisions by assembling packets from upper layers into long packets and short packets, and then uses a block-based lightweight FEC scheme to timely recover corrupted packets.
- A novel RSSI-based error detection mechanism to identify erroneous blocks with almost no overhead.
- Implementation and evaluation of the ACR prototype on a Tmote testbed. Results demonstrates that ACR achieves 25% higher transmission efficiency than existing recovery schemes.

The rest of the paper is organized as follows. After related work in Section II, we present empirical results from experiments that investigate partial packet error patterns due to collisions in section III. In section IV, we provide a theoretical study on the benefit of LS-collisions. In section V, we present the details of our ACR design and implementation. In section VI, we evaluate the performance in a real testbed. Finally, in section VII, we conclude the paper.

II. Related Work

In the state-of-the-art research, a significant number of MAC protocols have been proposed to deal with packet collision in WSNs. Basically, those MAC protocols can be divided into TDMA based protocol [8] [9], CSMA based protocols [2] [10], and hybrid protocols [4]. Among those protocols, TDMA based and hybrid protocols need synchronization, which brings extra overhead and complexity of clock synchronization. In addition, with time slots pre-assigned to nodes, these protocols cannot support bursty traffic efficiently. CSMA protocols can efficiently reduce collisions using clear channel assessment and randomized backoff, but intrinsically cannot eliminate all collisions, because of hidden terminal problems, as well as collisions when multiple nodes sense the medium free at the same time. Previous work [3] demonstrates that CSMA based protocols cannot avoid such collisions when the number of concurrent transmission grows.

Since collisions cannot be eliminated, packet recovery methods must be used. Traditional ARQ method uses ACK and retransmissions. However, ARQ brings high latency because of waiting for ACKs, and retransmissions cause more collisions, especially with heavy traffic. Partial Packet Recovery (PPR) methods exploit PCP, where only some bits are actually corrupted in a "lost" packet. In order to exploit PCP in WSNs, an adaptive FEC scheme is proposed in [5] to help receivers recover such partial corrupted packets by sending packets with redundant bits. In [11], packets are divided into blocks, and each block is attached with CRC codes. When packets are corrupted, only erroneous blocks are retransmitted. Both schemes reduce retransmissions and delivery latency. However, it is still unclear if such methods can be used to recover packets corrupted by collisions in WSNs.

Two novel partial packet recovery schemes have been recently proposed to recover corrupted packets from collisions. PPR protocol in [7] uses an asynchronous link-layer ARQ protocol that allows a receiver to compactly encode a request for retransmission of only those erroneous bits. Zigzag decoding in [6] proposes a new form of interference cancelation that exploits asynchrony across successive collisions caused by hidden terminals. However, both techniques need support from customized hardware. For example, PPR needs the hints from radio chips to detect which codeword is corrupted, and Zigzag implementation modifies modulation modules to recover bits from errors. They do not directly apply for the widely used off-the-shelf sensor devices like MicaZ and TelosB. Differently, our ARC actively changes the form of collisions to exploit the good PCP of LS-collisions, and can directly apply on almost any existing hardware. In this paper, we integrate ACR into off-the-shelf Tmote-Sky motes, and demonstrates great performance enhancement.

Previous work in [12] also utilizes different packet sizes to recover the collision. The difference is that, we actively set two packet lengths to create more LS-collisions, while they utilize the different packet lengths that already exist.

III. Empirical Measurement

In this section, we present our measurement of bit error patterns in collided packets using CC2420 radios on Tmote-Sky motes, to answer the following questions: 1) Can we apply PPR schemes to recover collided packets? 2) In what type of collisions will the collided bits be easy to recover?

A. Bit Error Pattern Comparison

We conduct a set of experiments on a testbed consisting of 48 Tmote-Sky motes to measure and compare bit error patterns in different cases. In the first set of experiments, we measure bit errors in corrupted packets due to poor link quality. We pick 5 pairs of senders and receivers, each of which is deployed in different rooms. The links between senders and receivers have poor qualities because of obstacles (walls), noise and multipath fading. Every sender transmits 20 packets per second for 10 minutes, each with a payload of 100 bytes. Senders use different frequencies to avoid collisions.

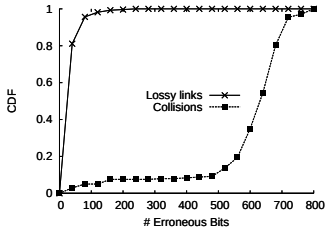


Fig. 1: CDF of # corrupted bits

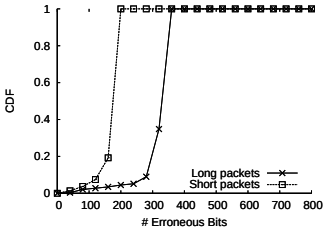


Fig. 2: CDF of # corrupted bits in long and short packets

With the CRC check disabled, receivers can collect corrupted packets. We repeat the experiments 5 times to get average results. In the second set of experiments, we study the bit error pattern in collided packets. We select 6 close nodes, among which we pick one node as the receiver and the others as senders. Every sender transmits packets in the same way as the first experiments. This time all senders use the same frequency and start to send at the same time to generate collisions. Senders use the default CSMA protocol in TinyOS 1.x. We also repeat the experiments 5 times.

Figure 1 plots the cumulative distribution of the number of erroneous bits in corrupted packets. It is clear that bit error patterns are very different for the two scenarios. In the lossy link case, around 80% of the corrupted packets have less than 50 erroneous bits, which is only 6.25% of the whole packet. On the contrary, over 60% of the collided packets have more than 600 erroneous bits, which is 75% of all bits. This observation indicates that PPR methods in [7] [11] can not achieve the same efficiency when recovering collided packets. First, since most bits are erroneous in collided packets, the performance gain of partial recovery is very limited. Second, since collisions cause more packet losses than corruptions, the chances to use partial recovery are limited. Actually, considering the cost and the complexity of the PPR implementation, it is more efficient to use the straightforward ARQ scheme for collision recovery.

Here, we also discuss why collisions generate such error patterns. In CSMA, nodes send packets only after sensing a free channel. Since all senders are close in our experiments, if one sender has been transmitting, others will wait. However, if two nodes sense a free channel at very close time points, they will send simultaneously and cause collisions. Such collisions produce two possible consequences. First, the Synchronization Headers (preamble and SFD) of both packets might be corrupted, which leads to packet loss at receivers. Second, even when a receiver synchronizes with one packet transmission, the signal of another packet may arrive shortly and start to corrupt the reception of the first one. Because two transmissions take almost the same time due to the same packet size, most bits of the first packet will be corrupted. CSMA already tries to eliminate such collisions, by making nodes randomly select different contention slots to sense the channel. However, when the number of transmission attempts grows, the likelihood of such collision increases. This collision can often be found in event-driven WSNs, or around sinks.

B. LS-Collision

Our discussion shows that partial packet recovery cannot be used to recover packets corrupted by collisions. Are there some techniques that result in collisions that do meet these criteria? We have found such a technique. In the third set of experiments, we use one receiver and 5 senders, among which one sends packets with a payload of 100 bytes (typical for multimedia & medical readings in health care applications), and other senders transmit packets with a payload of 25 bytes (typical for climate readings in environment monitoring applications). All the other settings are the same as the second set of experiments. We also repeat this experiment 5 times. Each time, we use a different sender to transmit long packets. We measure the distribution of the number of erroneous bits in collided packets, separated by long packets and short packets, as shown in Figure 2. First, around 80% of the collided short packets have over 150 erroneous bits, 75% of the whole packet. This is a similar pattern with that of collided packets in the second set of experiments, where most bits in packets are corrupted. However, long collided packets show different patterns. In this experiment, over 90% of the collided packets have 300 to 360 erroneous bits, while most collided packets in the second experiment have more than 600 erroneous bits. Also, the number of erroneous bits in long collided packets is similar to the size of short packets (368 bits). We repeat the experiment with different long and short packet sizes and different mixtures of long and short packets, and observe the same phenomena. We call such collisions *LS-collisions*, which has two necessary conditions: 1) collisions occur between long and short packets; 2) receivers synchronize with long packets transmissions.

A LS-collision has some good features. First, when a LS-collision occurs, a receiver still receives many correct bits in long collided packets. Second, the PCP in collided packets is predictable. The number of erroneous bits in collided packets is limited by the size of short packets, and most corrupted bits are continuous. Such features can help design efficient methods to recover long collided packets. For instance, we can design a FEC scheme to recover long packets. Since the number of erroneous bits is limited, it is easy to determine the appropriate degree of redundancy.

However, there are still challenging problems for exploiting LS-collisions. First, it should be justified whether exploiting LS-collisions can really improve the overall performance. Second, if LS-collisions are helpful, how can we have more LS-collisions instead of other collisions? How can we make nodes send different sizes of packets instead of the same size of packet as in most existing WSNs? How can we reduce collisions between long packets and between short packets? We will answer these questions in the following sections.

IV. Model and Analysis

A. System model

We begin by describing the system model in typical WSNs, where nodes send packets on one channel using CSMA. We

assume that at the beginning, there are N nodes attempting to send packets simultaneously and competing for transmissions. According to the CSMA implementation in TinyOS 1.x, the backoff time is divided into T contention slots. Each node picks a random contention slot $t \in [0, T]$. At contention slot t, the node carrier senses the medium. If the channel is free, it begins its transmission. Otherwise, it aborts or delays its transmission. If multiple nodes pick the same slot, they all sense a free medium and transmit at the same time, causing a collision. $P(t)$, $t = 0, 1, ..., T$, denotes the probability that a node picks the contention slot t. Obviously $0 \leq P(t) \leq 1$ and $\sum_{t=0}^{T} P(t) = 1$. We assume that each node independently selects the contention slot conforming to the same distribution. At last, D and H denote the length of the payload and the message header in packets, respectively. Next, we define two important performance metrics, the probability of successful transmission and the transmission efficiency.

Definition 1: A node wins slot r if and only if it is the only one choosing slot r, and all others choose later slots. There is a *successful transmission* if and only if some node wins some slot in $0, \ldots, T-1$. Let $\pi_p(N)$ be the probability of successful transmission when N nodes select a contention slot using probability distribution p.

Definition 2: Let r be the ratio of useful bits correctly received divided by all bits sent by senders. The transmission efficiency $E_p(N)$ is the expected r when N nodes select a contention slot using probability distribution p.

Two metrics we study here are crucial to the overall performance of a WSN. First, the probability of successful transmission can decide the channel throughput, and it can also impact the latency of packet delivery. A lower $\pi_p(N)$ means nodes wait for more competition rounds to win the transmission chance. Second, the transmission efficiency is an indicator whether nodes use transmission energy efficiently. In energy-constrained WSNs, higher transmission efficiency is always desired.

Next, we consider a CSMA model with LS-collision recovery, called LS-CSMA model. First, we assume that there are two types of nodes, one type transmitting long packets and the other type transmitting short packets. We call them *long nodes* and *short nodes*, respectively. Let ρ be the fraction of long nodes among all nodes. Thus, the numbers of long and short nodes are $N\rho$ and $N(1-\rho)$. Let $P_L(t)$ and $P_S(t)$ be the probability distributions on slots for long and short nodes, respectively. Also, we use D_L,H_L and D_S,H_S to denote the length of the payload and the message header of long packets and short packets. We assume that when one long node and multiple short nodes pick the same slot i and all others choose later slots, a LS-collision occurs. Further, we assume that a FEC scheme is applied and F redundancy bits are added at the end of long packets, which guarantees that long collided packets can always be recovered after LS-collisions. Based on above assumptions, the probability of successful transmission should be modified.

Definition 3: Under the LS-CSMA model, There is a *successful transmission* if and only if some node wins some slot, or a LS-collision occurs in some slot in $0, \ldots, T-1$. Let $\pi_p^{LS}(N)$ be the probability of successful transmission under LS-CSMA model. Also, let $E_p^{LS}(N)$ be the transmission efficiency under this model.

B. Performance gain from LS-collision

We now derive equations for $\pi_p(N)$ and $E_p(N)$ under a pure CSMA model. First, the probability of some node wins slot t is $NP(t)S(t+1)^{N-1}$, where $S(t+1) = \sum_{i=t+1}^{T} P(i)$ is the probability that the node selects any slot after slot t. Since the probability of successful transmission is the sum of all probabilities in each slot before slot T, we have

$$\pi_p(N) = \sum_{t=0}^{T-1} NP(t)S(t+1)^{N-1} \tag{1}$$

This equation is also given in [13][14]. For the transmission efficiency, since $r = D/(D+H)$ if successful transmission and $r = 0$ otherwise, we have

$$E_p(N) = \frac{D}{D+H} \sum_{t=0}^{T-1} NP(t)S(t+1)^{N-1} \tag{2}$$

Next, the following theorem states these two metrics under the LS-CSMA model.

Theorem 1: The probability of successful transmission and the transmission efficiency of LS-CSMA model are given by the following equations:

$$\pi_p^{LS}(N) = \sum_{t=0}^{T-1} \Big[N(1-\rho)P_S(t)S_S(t+1)^{N(1-\rho)-1} S_L(t+1)^{N\rho} + N\rho P_L(t)S_L(t+1)^{N\rho-1} S_S(t)^{N(1-\rho)} \Big] \tag{3}$$

$$E_p^{LS}(N) = \sum_{t=0}^{T-1} \Big[r_s N(1-\rho)P_S(t)S_S(t+1)^{N(1-\rho)-1} S_L(t+1)^{N\rho} + N\rho P_L(t)S_L(t+1)^{N\rho-1} \sum_{i=0}^{N(1-\rho)} r_i C_{N(1-\rho)}^{i} P_S(t)^i S_S(t)^{N(1-\rho)-i} \Big] \tag{4}$$

where $S_S(t+1) = \sum_{i=t+1}^{T} P_S(i)$ and $S_L(t+1) = \sum_{i=t+1}^{T} P_L(i)$, $r_s = D_S/(D_S+H_S)$, $r_i = D_L/(D_L+H_L+F+i*(D_S+H_S))$, $0 \leq i \leq N(1-\rho)$.

Interested readers can refer to the proof in [15]. These equations can be used to compare the performance of CSMA and LS-CSMA models. Also, the probability on contention slots can greatly impact values of these two metrics. As a first step, we use the uniform distribution for slots selection, which is commonly used in most CSMA protocols.

We compare the performance of CSMA and LS-CSMA, varying numbers of contenders. We assume that the numbers of long and short nodes are the same. Figure 3(a) shows successful transmission probabilities of the two cases, computed by Equation 3. First, it is clear that LS-CSMA always achieves higher success probabilities than CSMA under the same condition. For example, when 20 nodes compete for 8 contention slots, LS-CSMA can improve the success probability from 25.2% to 55.1% over CSMA. From another perspective, if we set an acceptable threshold for the success probability, such as 70%, with 16 contention slots, CSMA can only contain around 10 nodes competing for transmissions, but

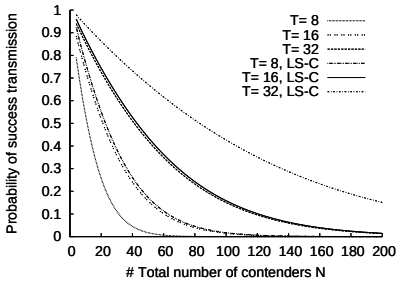


Fig. 3: CSMA vs. LS-CSMA

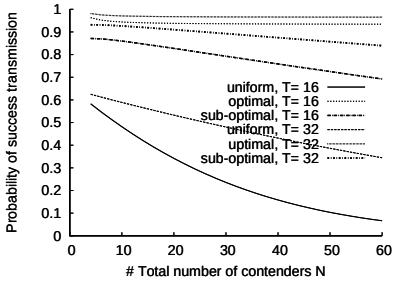


Fig. 4: Compare uniform, optimal and near-optimal distributions

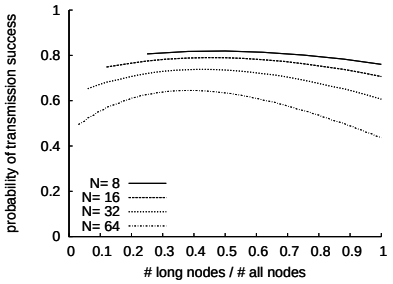


Fig. 5: Proportion of long and short nodes

LS-CSMA can support 24 nodes and still meet the threshold. Second, LS-CSMA even slightly outperforms CSMA with double contention slots, such as LS-CSMA with $T = 8$ and CSMA with $T = 16$, as well as LS-CSMA with $T = 16$ and CSMA with $T = 32$. This observation indicates that LS-collision recovery can help CSMA achieve the same, or even higher success probability by using half of the number of slots. Above all, LS-CSMA can provide more reliable transmission, alleviate transmission contention, and reduce packet delivery latency by using much fewer contention slots. We also study the transmission efficiency of the two models. Similar to previous results, LS-CSMA always achieves higher transmission efficiency. For example, with 20 nodes competing in 16 slots, LS-CSMA improves efficiency from 0.34 to 0.43 over CSMA. Interested readers can refer to the result in [15].

With a common uniform distribution, LS-CSMA outperforms CSMA in both metrics. However, as shown in [13][14], the uniform distribution is not the optimal slot selection strategy for CSMA. Performance can be further improved by using better distributions.

C. Slot Selection Distribution

We first describe how to optimize the probability of successful transmission $\pi_p^{LS}(N)$, according to Equation 3. We assume that all variable values are given, except the distributions $P_L(t)$ and $P_S(t)$. The goal is to find the optimal distribution for both long and short nodes. Here, we use a two-step method. This two-step method comes from one observation. In order to achieve higher successful transmission, the thing we want to eliminate the most is that multiple long packets collide in slot t, because in that case, no matter what short nodes select, there is no successful transmission. Since long and short nodes pick slots independently, our first step is to find the optimal $P_L(t)$ which minimizes the probability that multiple long nodes collide in any slot, without considering short nodes' behavior. Let P_L^* denote this optimal distribution. For the second step, given the probability of long nodes, we can find the optimal distribution of short nodes, denoted by P_S^*.

Theorem 2: The optimal slot distributions of long and short nodes are given recursively as follows. Let $K_L^*(t) = \frac{\sum_{i=t}^{T} P_L^*(i)}{\sum_{i=t-1}^{T} P_L^*(i)}$ and $K_S^*(t) = \frac{\sum_{i=t}^{T} P_S^*(i)}{\sum_{i=t-1}^{T} P_S^*(i)}$, where $1 \le t \le T$. We have

$$K_L^*(t-1) = \frac{N_l - 1}{N_l - K_L^*(t)^{N_l - 1}} \tag{5}$$

$$K_S^*(t-1) = \frac{N_s - 1}{N_s - K_L^*(t)^{N_l - 1}\left[N_l + K_L^*(t)(K_S^*(t)^{N_s - 1} - N_l)\right]} \tag{6}$$

where $t = 2, ..., T$, $N_l = N\rho$, $N_s = N(1-\rho)$ are the numbers of long and short nodes. Also, $K_L^*(T) = \frac{N_l - 1}{N_l}$ and $K_S^*(T) = \frac{N_s - 1}{N_s}$.

Interested readers can refer to the proof in [15]. With the above theorem, we can compute the optimal distributions for long and short nodes. However, we find it is hard to find the optimal distribution to maximize transmission efficiency. We can only use the same strategy to compute a sub-optimal distribution, which can also be found in [15].

Computing the optimal distribution requires the knowledge of the numbers of long and short nodes. These numbers may vary from time to time, especially in event-driven WSNs. Also, the process of computing the distribution is very complicated and too costly for power-limited sensor nodes. Therefore, we seek alternative distributions which provide sub-optimal, but simple solutions without the need for exact information about other contenders.

Table 2 shows some examples of the optimal distribution. First, we can see optimal distributions of long and short nodes are very similar. Both probabilities increase slowly in the first few slots, but raise rapidly in the last few slots, especially when the number of nodes is large. Such a distribution reduces the number of nodes that pick previous slots and thus improves the probability of a successful transmission. Second, comparing $P_S^*(t)$ with $P_L^*(t)$, we find that in most cases, the probability of short packets is slightly less than that of long packets in all first $T-1$ slots, and the probability of short packets is more concentrated in the last slot. The effect of this is to avoid the situation that multiple short packets pick slot t and all long packets pick later slots. The first observation suggests using an increasing geometric sequence to compute a sub-optimal distribution. [13][14] also use the geometric sequence to optimize the success probability of CSMA. [14] gives a distribution candidate, in which

$$P(t) = \frac{b^{\frac{t+1}{T+1}} - b^{\frac{t}{T+1}}}{b-1} \tag{7}$$

where $t = 0, ..., T$, and b is a number greater than 1. In this geometric sequence, the increasing ratio is $b^{\frac{t}{T+1}}$. Here

		$P^*(0)$	$P^*(1)$	$P^*(2)$	$P^*(3)$	$P^*(4)$	$P^*(5)$	$P^*(6)$	$P^*(7)$	$P^*(8)$	
$N = 16$	Long node	0.02651	0.02898	0.03206	0.03607	0.04153	0.04952	0.06269	0.09033	0.63229	$\pi_p^{LS}(16) = 0.89$
	Short node	0.01752	0.01951	0.02207	0.02553	0.03052	0.03844	0.05351	0.09911	0.69380	
		$P^*(0)$	$P^*(1)$	$P^*(2)$	$P^*(3)$	$\cdots$	$P^*(29)$	$P^*(30)$	$P^*(31)$	$P^*(32)$	
$N = 64$	Long node	0.00184	0.00190	0.00195	0.00201	$\cdots$	0.01303	0.01731	0.02677	0.82997	$\pi_p^{LS}(64) = 0.967$
	Short node	0.00116	0.00119	0.00123	0.00127	$\cdots$	0.00989	0.01426	0.02814	0.87242	

TABLE I: Examples of the Optimal Distribution. (TOP) Optimal Distribution for $T = 8$. (BOTTOM) $T = 32$

we use a small value of b, where $b = 10$ for long nodes, and $b = 12$ for short nodes. According to the second observation above, short nodes use a larger b than long nodes. We compare the uniform, optimal and sub-optimal distribution by varying the number of contenders. Results are shown in Figure 4. We can see that optimal and sub-optimal distributions always achieve better performance than uniform the distributions in every case. When comparing sub-optimal distribution with the optimal, we can see that the sub-optimal result is very close to the optimal result with 32 contention slots. For example, with 20 nodes, its ratio to the optimal result is 0.93. When contention slots are less than 32, this sequence can still work well, as the number of contenders is not too high. Consider that the number of senders in one neighborhood is usually less than 30 in most WSNs, the geometric sequence can still give us good performance in most cases. Furthermore, it can be simply implemented by computing the slot i with the following equation:

$$i = \lfloor (T+1)\log_b[\alpha(b-1)+1] \rfloor \tag{8}$$

where α is a random variable with a uniform distribution within the interval $[0, 1)$, and $b = 10$ for long nodes or $b = 12$ for short nodes.

The last problem is how to determine whether a node should be a long or short node. The basic idea works as follows. When a node has messages to send, it will "toss" the coin to make the decision, according to a pre-defined probability. In other words, we need to determine the right value of ρ, to optimize the two metrics. To this end, we compute the probability of successful transmission, with different values of ρ. Here, we use the geometric sequence discussed above to generate the distribution on 16 slots. Results are shown in Figure 5. We can see that the success probability changes smoothly while ρ changes. All maximum values appear in the middle of the curve, which means the best value of ρ is around 0.5. That makes sense, because if unbalanced, nodes of the majority are likely to generate more collisions among themselves. Such long-long, or short-short collisions will cause transmission failures and are not preferred in the LS-CSMA model.

V. ACR DESIGN

In this section, we present the design of ACR, the *Active Collision Recovery* protocol, which exploits the advantages of LS-collisions to efficiently recover collided packets. ACR is implemented as a link layer protocol combined with a slight modification on CSMA protocols. Figure 6 illustrates the functions of ACR at both the sender and the receiver sides.

A. ACR Sender

A sender node of ACR actively converts potential collisions into LS-collisions and then applies hybrid schemes to recover corrupted packets from collisions. At senders, the ACR protocol has the following steps:

1) Packet Assembly: One key condition for LS-collision is the co-existence of long and short packets. Thus, the Packet Assembly (PA) step actively assembles packets from the network layer into long packets or short packets, and feeds them into the MAC layer. PA needs to determine if nodes should send long or short packets, by "tossing a coin" with the probability ρ. Here, we use 50%, according to the analysis in Section IV-C. Another important problem is to select the right sizes for long and short packets, which depends on different radio hardware and collision conditions. In this paper, we focus on off-the-shelf mote platforms, such as MicaZ or Tmote-Sky, which can send a packet as long as 128 bits. We select packet sizes of 25 and 80 bits for short and long packets.

2) FEC and CRC: If nodes are to send short packets, ACR adds a CRC code at the end of each short packet. Later, receivers use this CRC as a checksum to verify the integrity of the packet. In the implementation we apply a 16-bit CRC. If nodes are to send long packets, ACR uses FEC codes to allow receivers to recover long packets without retransmission when LS-collisions occur, which reduces communication overhead as well as recovery latency. The basic idea is to divide a long packet into n blocks, and then compute m redundant blocks attached at the end of packets, so that a receiver can still reconstruct the original packets until more than m blocks are corrupted. One traditional way is to compute $n + m$ linear combinations of the original n blocks, and assemble these $n+m$ linear combinations to a packet to transmit. Any correct n combinations are sufficient for receivers to reconstruct the original data. However, such a method requires a large amount of matrix operations, not suitable to Tmote nodes. Instead, we take advantage of the special error patterns of LS-collisions to design a lightweight FEC scheme. Figure 7 shows an example of how to encode a packet. Here, a packet is divided into n blocks, $d_0 \cdots d_{n-1}$. m redundant blocks are computed by

$$C_{0,j} = d_{0,j} \oplus d_{1,j} \oplus \cdots \oplus d_{n-1,j},$$

$$C_{k,j} = d_{k,j} \oplus d_{k+m,j} \oplus \cdots \oplus d_{k+m\cdot\lfloor\frac{n-1}{m}\rfloor,j},$$

where $0 \le j \le t$, $1 \le k \le m-1$ and $\oplus$ denotes Exclusive OR operations. As observed in Section III, the number of corrupted bits in collided packets is not greater than the size of short packets, denoted by S. Thus, we set the size of redundant bits $(t+1)m >= S$, which guarantees that redundant bits are always more than corrupted bits. For the

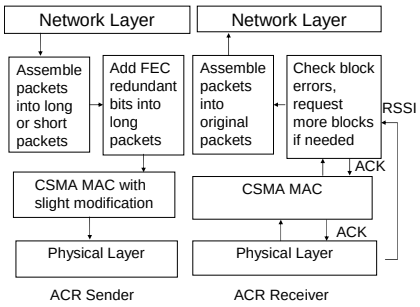


Fig. 6: ACR components

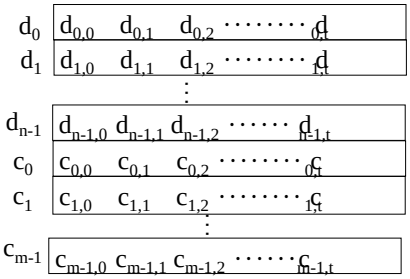


Fig. 7: Illustration of FEC encoding

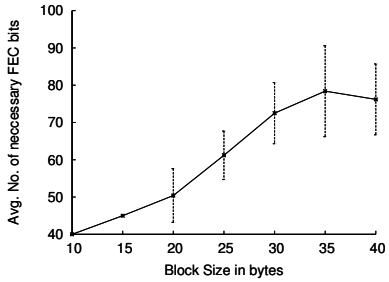


Fig. 8: Smaller block is more efficient

second observation, corrupted bits are continuous in collided packets, therefore those bits corrupt at most m continuous blocks, $d_j \cdots d_{j+m-1}$. Our encoding method guarantees that any two blocks within these m continuous blocks are not both used to generate one redundant block, except C_0. Therefore, this FEC method can always recover collided packets due to LS-collisions. This FEC method only requires bit operations for encoding and decoding, and requires no extra storage. Thus it can be implemented easily in sensor nodes.

Another problem is to determine the right size of blocks. We apply various sizes of blocks to collided packets in a real trace file, and measure the number of redundant bits (block_size $\times$ No. redundant block) necessary for recovery, as shown in Figure 8. Generally, the smaller blocks are, the less redundancy we need. The reason is that larger blocks are likely to cause more "wasted" bits, which are correct, but in corrupted blocks. As a result, smaller blocks are preferred. The size of blocks also depends on the method to identify erroneous blocks. As in [11], one extra byte of checksum is attached to each block for error detection. With smaller sizes of blocks, more blocks introduce more overhead. Therefore, the block size is bounded within 20$\sim$25 bytes in [11]. However, ACR applies a novel method for error detection, which efficiently identifies erroneous blocks without extra checksums, thus supports even smaller blocks. With this method, we use a block as small as 10 bytes. We will introduce this method later.

After adding the FEC codes, we also add 16 bits of CRC code at the end of packets, which are computed based on the original packets without FEC bits.

3) Modified CSMA: ACR slightly modifies a CSMA protocol to further increase the chances of LS-collisions. There are two changes.

Non-uniform Distribution on contetion slots. Even with long packets and short packets, contenders can still end up with collisions other than LS-collisions, such as collisions between long packets. According to Section IV-C, we can use Equation 8 to compute a non-uniform distribution for senders, which increases the likelihood of LS-collisions and results in better performance. This only requires a minor change on the underlying MAC layer (such B-MAC, S-MAC), replacing the uniform backoff distribution of those protocols.

Transmission Delay on short packets. Another necessary condition for LS-collision is that receivers should get synchronized with long packets' transmission first. In order to realize this condition, we introduce a slightly delay on short packets' transmissions. When sensing a channel clear, short packets wait for a time τ, while long packets do CCA immediately. Here, τ is equal to the time to transmit the synchronization header of a packet, which is roughly 130 μs in CC2420 radios. This time is shorter than one contention slot (e.g. 320μs in TinyOS), thus it does not impact CSMA protocols.

4) Backup ARQ scheme: At last, ACR also provides a backup ARQ scheme to recover lost packets in three cases. 1) Packets lost due to other kinds of collisions; 2) short packets lost in LS-collisions; 3) long packets that have no sufficient correct blocks to be recovered by the FEC scheme. In this ARQ scheme, a sender waits for an ACK after a transmission. If the ACK is not received after a timeout, it retransmits the previous packet. If the ACK is received with notification that several blocks are needed for recovery, it retransmits that number of blocks, instead of the whole packet. This ACK scheme brings extra overhead by retransmission. But since the FEC scheme can recover most long collided packets from LS-collisions, the number of retransmissions is less than that of the pure ARQ.

B. ACR Receiver

When receiving packets, ACR first determines if they are short or long packets. If it is short, ACR uses the attached CRC to determine if the packet is correct or not. If the CRC check is passed, the receiver sends an ACK message.

If nodes receive long packets, ACR also uses the CRC to determine whether there are any erroneous blocks in this long packet. If the packet passes the CRC check, receivers send a success ACK. Otherwise, receivers know some blocks are corrupted. Next, ACR needs to identify erroneous blocks. As mentioned before, ACR does not add extra bits on each block to detect corruption, because it costs extra overhead. Instead, we introduce a new method. This new method is based on the fact that during reception, RSSI values at the receiver side increase when collisions occur, but are almost constant without collision. We conduct experiments to measure the RSSI value in the course of packet reception. One typical result is shown in Figure 9. We can clearly see that the RSSI value increases to above $-68dBm$ when collisions corrupt blocks. Without collisions, the RSSI value is stable at $-73dBm$. Thus, we can detect a corrupted block by checking

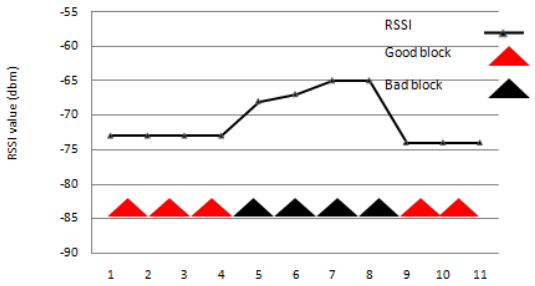


Fig. 9: Relation between RSSI and bad blocks

whether the RSSI value during the reception of this block is high or normal. In our method, a receiver continuously samples RSSI values every δ time units and keeps them in an array, as soon as a reception begins. Later after the reception is over, ACR looks up the array to seek the instant increase of RSSI values. If it finds any high value, it maps the value to the corresponding block, and marks it as a potential corrupted block. Here time δ should be equal to the time of receiving one block. We implement this RSSI based error detection method in Tmote nodes, and find that the maximum RSSI sampling rate that hardware can support is one reading per $200{\sim}250\mu s$, which means that this method can support the block as small as 8 bytes without extra overhead. Considering other factors, we choose the block size as 10 bytes. By using this method, we can identify most corrupted blocks in our experiments. Occasionally, this method missed some bad blocks at the beginning of the collision. Then ACR cannot recover the long packet and it uses the backup ARQ to recover them.

After erroneous blocks are found, ACR first checks if the number of erroneous blocks is greater than m. If so, the FEC scheme cannot recover the blocks, and ACR sends an ACK message with a block bitmap to request senders to send good blocks. Note that it is not necessary to send all erroneous blocks, instead ACR only needs good blocks until the number of erroneous blocks is not greater than m. With less than $m+1$ erroneous blocks, ACR use the FEC codes to recover these blocks. For a corrupted block d_k, suppose $k = i \cdot m + r$, if $r \neq 0$, then this block are reconstructed by

$$d_{k,j} = c_{r,j} \oplus d_{r,j} \cdots \oplus d_{(i-1)m+r,j} \oplus d_{(i+1)m+r,j} \cdots \oplus d_{\lfloor \frac{n-1}{m} \rfloor m+r,j},$$

If $r = 0$,

$$d_{k,j} = c_{0,j} \oplus d_{0,j} \cdots \oplus d_{k-1,j} \oplus d_{k+1,j} \cdots \oplus d_{m-1,j},$$

where $0 \leq j \leq t$. Here, we should always compute blocks with $r \neq 0$ first, and them reconstruct the block with $r = 0$. After recovering corrupted blocks, ACR conducts the CRC check, and sends an ACK if it passes the CRC check.

C. Discussion

ACR is designed to deal with collisions when multiple nodes are sending packets simultaneously. As mentioned before, this scenario occurs frequently in dense WSNs, when events trigger burst-traffic, or when high-volume traffic is aggregated at sinks. However, when there are few collisions, using ACR brings extra overhead. Therefore, it is better to combine CSMA and ACR into an adaptive protocol, which applies CSMA when there are few collisions, but switches to ACR when multiple nodes start to compete for transmissions. This adaptive protocol can be a promising protocol to achieve efficiency under various scenarios.

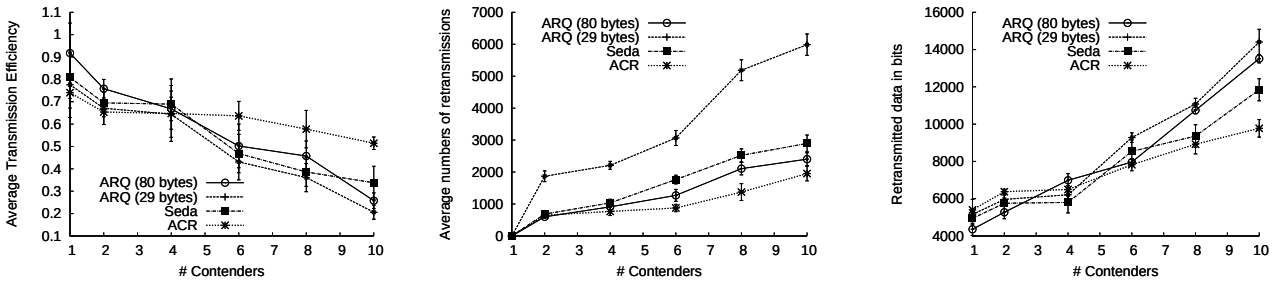
The second concern is the efficiency of packet assembling. People may argue that assembling packets may bring overhead by segmenting large packets into small ones. But in current WSNs, most packets tend to be small, so most assembling actions are to aggregate small packets into large ones. Furthermore, our analysis in Section IV shows that ACR achieves better transmission efficiency than CSMA.

Another design concern is energy efficiency. Energy consumption in WSNs primarily comes from idle listening. ACR does not directly focus on how to reduce the idle listening, but ACR is compatible with MAC protocols that address this issue, such as B-MAC [1], mainly because ACR does not demand any nodes to overhear or receive extra signals except receivers.

VI. Performance Evaluation

In this section we evaluate the performance of ACR on a physical testbed. The main scenario we consider is that multiple nodes have a certain amount of information to transmit to one sink. In our testbed, all nodes can send packets to the sink in one hop. We compare ACR with two other types of packet recovery schemes, pure ARQ and Seda [11]. In Seda, a long packet is divided into blocks. Receivers send ACKs to request senders to retransmit corrupted blocks. Unlike ACR, Seda adds a 1-byte CRC and a 1-byte sequence number to each block for error detection. Default values used in our experiments are as follows. Every sender transmits at a power of $-10dBm$, which ensures good links. The senders have 400 bits of information to send every second. We consider two packet sizes for pure ARQ, namely 29 and 80 bytes. For Seda, the data size of one packet is 80 bytes, divided into 4 blocks, each of which has 20 bytes of data, as suggested in [11]. For ACR, short packets have 25 bytes of data, and long packets have 80 bytes data, divided into 8 blocks, with a block size of 10 bytes. It also carries 4 blocks of redundancy bits. For all three methods, we shorten the MAC header, and make the length of the whole header 6 bytes. Also, the ARQ times out after $2ms$. At last, all methods use the default CSMA protocol in TinyOS 1.x, with 16 contention slots. All experiments run for 10 minutes and send 4000 bits. Each data point is obtained by averaging results of 5 trials. We also add 90% confidence intervals for each data point.

Figure 10(a) plots the measured transmission efficiency of all methods with various numbers of senders, calculated by the information bits transmitted over all bits transmitted or retransmitted. We observe that ACR has the lowest efficiency when there is only one contender with no collisions, because the long packets of ACR have extra redundant bits. However, when the number of contenders grows, ACR starts to recover long collided packets from LS-collisions and outperforms



(a) Transmission Efficiency (b) # Retransmissions (c) Volume of data retransmitted

Fig. 10: Recovery performance under various # contenders

other schemes. This result implies that ACR is more efficient to recover collisions than other methods. For example, with 6 contenders, ACR improves by ∼25% the transmission efficiency over ARQ. On the contrary, Seda has the lowest efficiency when collisions increase. This is mainly because collided packets do not have a partial packet pattern, so Seda needs retransmit the whole packet. Further, Seda pays extra overhead of checksums.

Figure 10(b) plots the number of retransmissions with various number of senders. We can see that with heavy collisions, ACR triggers much fewer retransmissions than others. For example, with 8 contenders, ACR reduces retransmissions by 42% over ARQ with long packets. This reduction comes from two places. First ACR applies a better distribution to select contention slots, which leads to fewer collisions. Second, ACR generates LS-collisions, and uses a FEC method to recover long collided packets from such collisions, thus it only needs to retransmit short collided packets. Other methods have to retransmit all collided packets after collisions.

Figure 10(c) plots the number of bits retransmitted with various number of senders. This figure demonstrates that with heavy collisions, ACR retransmits much fewer bits than others. For example, with 10 contenders, ACR reduces retransmitted bits by 17.4% over Seda. As discussed before, the main reason of this reduction is that ACR recovers most long packets without retransmission, therefore it saves a large number retransmitted bits. ACR also achieves higher probabilities of successful transmission and overall throughput, shown in [15].

VII. Conclusion

This paper explores solutions to efficiently recover collisions in WSNs. Through empirical experiments, we first discover that LS-collisions have exploitable partial corrupted patterns in collided packets. Then, we propose ACR that actively converts potential collisions into LS-collisions and also uses lightweight FEC to recover collided packets from LS-collisions. A novel RSSI-based error detection method is also developed in ACR. We present theoretical analysis that shows that ACR greatly improves the probability of successful transmissions and enhances the transmission efficiency. We evaluate ACR performance in TinyOS with Tmote-Sky motes, and we demonstrate that ACR is more efficient in recovering collisions than existing solutions, by significantly reducing the number of retransmissions and increasing the transmission efficiency.

In the future, we plan to investigate ACR's performance in multi-hop WSNs, and further improvce efficiency by using good adaptive mechanisms that add FEC bits only when the collision is often.

VIII. Acknowledgments

This work has been supported, in part, by NSF grants CNS-0615063, CNS-0614870, CNS-0626632 and ECCS-0901437.

References

[1] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Median Access for Wireless Sensor Networks," in *ACM SenSys*, 2004.

[2] M. Buettner, G. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *ACM SenSys*, 2006.

[3] K. Jamieson, H. Balakrishnan, and Y. C. Tay, "Sift: A mac protocol for event-driven wireless sensor networks," Tech. Rep., MIT Laboratory for Computer Science, 2003.

[4] G.-S. Ahn, E. Miluzzo, A. T. Campbell, S. G. Hong, and F. Cuomo, "Funneling-MAC: A Localized, Sink-Oriented MAC for Boosting Fidelity in Sensor Networks," in *ACM SenSys*, 2006.

[5] J. Ahn, S. Hong, and J. Heidemann, "An adaptive fec code control algorithm for mobile wireless sensor networks," *Journal of Communications and Networks*, 2005.

[6] S. Gollakota and D. Katabi, "Zigzag decoding: combating hidden terminals in wireless networks," in *ACM SIGCOMM*, 2008.

[7] K. Jamieson and H. Balakrishnan, "Ppr: Partial packet recovery for wireless networks," in *ACM SIGCOMM*, 2007.

[8] S. Mishra and A. Nasipuri, "An adaptive low power reservation based mac protocol for wireless sensor networks," in *IEEE IPCCC*, 2004.

[9] Y. Sun, S. Du, O. Gurewitz, and D. B. Johnson, "Dw-mac: a low latency, energy efficient demand-wakeup mac protocol for wireless sensor networks," in *ACM MobiHoc*, 2008.

[10] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A component-based architecture for power-efficient media access control in wireless sensor networks," in *ACM SenSys*, 2007.

[11] R. K. Ganti, P. Jayachandran, H. Luo, and T. F. Abdelzaher, "Datalink streaming in wireless sensor networks," in *ACM SenSys*, 2006.

[12] J. Manweiler, N. k. Santhapuri, S. Sen, R. R. Choudhury, S. Nelakuditi, and K. Munagala, "Order matters: Transmission reordering in wireless networks," in *ACM MobiCom*, 2009.

[13] Y.C. Tay, Kyle Jamieson, and Hari Balakrishnan, "Collision-Minimizing CSMA and its Applications to Wireless Sensor Networks," *IEEE Journal on Selected Areas in Communications*, 2004.

[14] G. Zhou, C. Huang, T.Yan, T. He, and J. A. Stankovic, "MMSN: Multi-Frequency Media Access Control for Wireless Sensor Networks," in *IEEE Infocom*, 2006.

[15] Y. Wu and J. A. Stankovic, "Collided Packet Recovery in Wireless Sensor Networks," Tech. Rep., http://www.cs.virginia.edu/∼yw5s/ACR.pdf.