Chase: Taming Concurrent Broadcast for Flooding in Asynchronous Duty Cycle Networks

Zhichao Cao*, Jiliang Wang*, Daibo Liu†, Xiaolong Zheng*
*School of Software and TNLIST, Tsinghua University, China
†School of CSE, University of Electronic Science and Technology of China
{caozc, jiliang, xiaolong}@greenorbs.com, dbliu.sky@gmail.com

***Abstract*— Asynchronous duty cycle is widely used for energy constraint wireless nodes to save energy. The basic flooding service in asynchronous duty cycle networks, however, is still far from efficient due to severe packet collisions and contentions. We present *Chase*, an efficient and fully distributed concurrent broadcast layer for flooding in asynchronous duty cycle networks. The main idea of *Chase* is to meet the strict signal timing and strength requirements (e.g., Capture Effect) for concurrent transmission while reducing contentions and collisions. We propose a distributed random inter-preamble packet interval adjustment approach to constructively satisfy the requirements. Even when requirements cannot be satisfied due to physical constraints (e.g., the difference of signal strength is less than a 3 dB), we propose a light-weight signal pattern recognition based approach to identify such a circumstance and extend radio-on time for packet delivery. We implement *Chase* in TinyOS and TelosB platform and extensively evaluate its performance. The implementation does not have any specific requirement on the hardware and can be easily extended to other platforms. The evaluation results also show that Chase can significantly improve flooding efficiency in asynchronous duty cycle networks.**

I. Introduction

Internet of things (IoT) [20] [18] [27] is becoming a promising way to enhance our daily life. Many battery powered wireless nodes in IoT have limited power supply. To save energy, low duty cycle radio management is widely used as radio is a major part for energy consumption [8]. Asynchronous duty cycle, namely LPL (Low Power Listening) (e.g., BoxMAC [21], Zisense [28]), is one of the most commonly used low duty cycle modes [14] [2]. In LPL, instead of keeping radio always-on, each node periodically turns on the radio to detect potential signal by sampling the received signal strength (RSS). If a signal is detected, the node keeps the radio on to receive potential incoming packet. Otherwise, the node turns off its radio and sleeps for a certain time period (called *sleep interval*). In LPL, nodes are not synchronized and have different schedules to turn on their radio (called *active schedule*).

Flooding is a fundamental service and a basic building block for LPL networks. For example, flooding is the basis for propagating messages (such as binary image [3][5] [22] [29], time stamp [19] [7], etc) to all nodes. Flooding is also widely used to notify nodes [16] and update system parameters [26], which is common for practical networks.

In LPL, each node keeps transmitting the same packet (called *preamble packet*) for whole sleep interval in order to ensure that the receiver can wake up once. In flooding, multiple nodes may simultaneously broadcast the same packet. Thus those nodes will keep transmitting the preamble packets at the same time, which introduces much more packet contentions and collisions than that in traditional networks [1]. This further leads to a dilemma for flooding in LPL. On one hand, a node needs to transmit the preamble packet multiple times to ensure the packet can be delivered. On the other hand, transmitting preamble packets from multiple nodes at the same time results in packet contentions and collisions. Consequently, as observed in Section II, the delay of flooding in LPL is very large.

Despite of the practical prevalence of LPL, the basic LPL network flooding is still not efficient. As a result, network protocols and services (e.g., network parameter update, network notification) built on top of flooding are also not efficient. In this paper, we present *Chase*, an efficient and fully distributed broadcast layer for LPL network flooding. The basic idea of *Chase* is to remove the influence of packet contentions/collisions and improve the concurrency of preamble packets, i.e., improve the successful ratio while multiple nodes are transmitting the preamble packets. Concurrent transmission has strict requirement on timing and signal strength. For example, for capture effect which enables concurrent transmission, the strongest signal should not arrive later than a certain tiny time offset after the first weak signal. Besides, the strongest signal strength is at least 3 dB larger than the sum of others.

In *Chase*, we design a random *Inter Preamble Packet Interval* (IPPI) adjustment technique to address the timing requirement. We analytically show that different nodes can achieve concurrent transmission with a high probability. Even when the requirements of concurrent transmission cannot be satisfied due to physical constraints, *Chase* leverages a light-weight signal pattern recognition based approach to identify such a circumstance and extend radio-on time to ensure packet delivery.

We implement *Chase* in TinyOS with TelosB nodes. The implementation does not have any specific requirement on the hardware and can be easily extended to other platforms. We conduct extensive evaluation and the evaluation results show that *Chase* can significantly improve the flooding performance in asynchronous networks. Our contributions are summarized as follows.

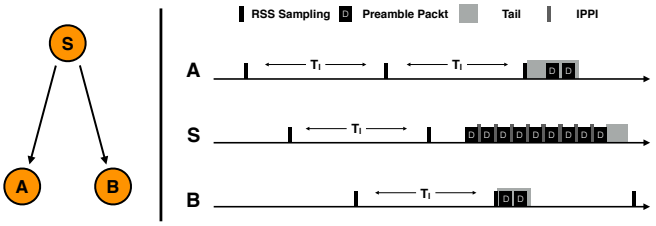
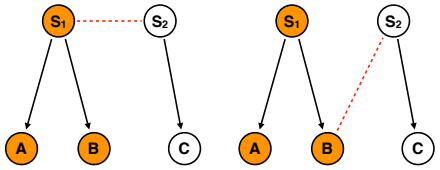


Fig. 1: Illustration of LPL broadcast from sender **S** to its neighbors **A** and **B**.



(a) Packet mishear (b) Packet collision

Fig. 2: Illustration of the situation of packet mishear and packet collision.

- We propose *Chase*, an efficient and fully distributed broadcast layer to support concurrent transmission for flooding in asynchronous duty cycle networks.
- We address the difficulties in supporting concurrent transmission in practical asynchronous duty cycle networks and design light-weight and efficient countermeasures.
- We implement *Chase* in TinyOS with TelosB nodes [24]. The evaluation results show that *Chase* can significantly improve flooding efficiency.

The rest of paper is organized as follows. Section II illustrates the basic model of LPL broadcast and performance of LPL flooding. Section III shows the detailed design of *Chase*. Section IV and V show the implementation and evaluation results, respectively. Section VI introduces the related works. Finally, Section VII concludes this work.

II. Empirical Study

In this section, we analyze the performance of flooding in LPL networks, and conduct experiments to show the inefficiency of flooding in LPL networks.

A. Flooding in LPL

As shown in Figure 1, **S** broadcasts packets to two neighbors **A** and **B**. The sleep interval is T_l for both **A** and **B**. In LPL, the active schedule are asynchronous. After turning on the radio, a node continuously samples the RSS for a time period of T_s (called *RSS Sampling*). Sometimes, the radio is further kept on for T_t (called *Tail*) to receive potential preamble packets. When **S** prepares to broadcast, **S** turns on the radio and keeps transmitting the preamble packets for a time period of T_p. The Inter Preamble Packet Interval between two consecutive preamble packets (i.e. IPPI) is denoted as T_{ippi}. The on air time of a preamble packet is denoted as T_a.

There are two requirements for broadcast in LPL. First, $T_p > T_l$ to ensure **S** can meet **A** and **B** at their rendezvous. Thus, broadcast in LPL will occupy the channel for a long time. Second, $T_s > T_{ippi}$ to ensure **A** and **B** can detect the signal from **S**. In practice, T_t is usually several times of T_a to ensure **A** and **B** can successfully receive at least one preamble packet.

The impact of flooding in LPL is from the following aspects. First, flooding in LPL will result in backoff for preamble packets, which further leads to increase of IPPI T_{ippi}. As we have mentioned, the sampling time T_s is related to T_{ippi}. When T_{ippi} increases, the sampling time T_s becomes less than T_{ippi}. As shown in Figure 2(a), $\mathbf{S}_1$ and $\mathbf{S}_2$ are broadcasting. $\mathbf{S}_2$ takes random backoff when it detects signal from $\mathbf{S}_1$. The backoff can significantly increase T_{ippi}. Thus T_s of **C** becomes less than T_{ippi}. Then **C** might fail to detect the signal from $\mathbf{S}_2$. We call such a phenomenon *Packet Mishear*.

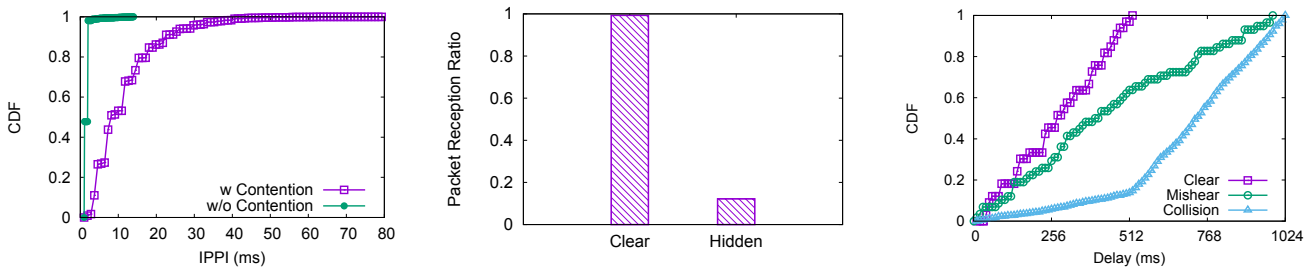
Second, it is even worse when two or more nodes cannot hear from each other. They may simultaneously transmit preamble packets which further results severe collisions. As shown in Figure 2(b), $\mathbf{S}_1$ and $\mathbf{S}_2$ are hidden terminals. Therefore, preamble packets from $\mathbf{S}_1$ and $\mathbf{S}_2$ collide sequentially. $\mathbf{S}_1$ must rebroadcast till **B** successfully receives one preamble packet.

B. Impact of Mishear and Collision

We further conduct experiments to show the impact of packet mishear and packet collision in real networks. As shown in Figure 4, we use the default settings of Box-MAC [21] in TinyOS. T_p is set to 532ms, which is 20ms larger than T_l. T_s is set to 12ms. The length of each preamble packet is set to 77 bytes and the corresponding on-air time is 2624 μs. We use channel 26 in the experiments, the least overlapped channel with interference (e.g. WiFi, Bluetooth).

First, we use two TelosB nodes $\mathbf{S}_1$ and $\mathbf{S}_2$ as shown in Figure 2(a). We first record T_{ippi} of $\mathbf{S}_2$ when only $\mathbf{S}_2$ broadcasts. We also record T_{ippi} of $\mathbf{S}_2$ when both $\mathbf{S}_1$ and $\mathbf{S}_2$ broadcast. The distribution of T_{ippi} is shown in Figure 3(a). For the first case, over 99.9% of T_{ippi} is less than T_s. However, for the second case, about 67.7% of T_{ippi} is less than T_s. Thus the probability of packet mishear is significantly increased. Even **C** detects the signal from $\mathbf{S}_2$, about 14% of T_{ippi} is larger than T_t so that **C** may not receive any preamble packet in tail.

In the second experiment, the topology is shown in Figure 2(b). We set the power level of $\mathbf{S}_1$ and $\mathbf{S}_2$ to 7. The average RSS difference between $\mathbf{S}_1$'s and $\mathbf{S}_2$'s signals is about 2 dB for **B**. The capture effect does not work in most of cases. We separately measure the packet reception ratio of link $\mathbf{S}_1 \rightarrow \mathbf{B}$ under clear environment and hidden terminal. As shown in Figure 3(b), almost all packets can be successfully received under clear environment. Under hidden terminal, although **B** can receive a few of packets, the packet reception ratio dramatically decreases to 12.2%.



(a) T_{ippi} distribution (b) Packet Reception Ratio (c) Delay distribution

Fig. 3: The impact of channel contention and hidden terminal on T_{ippi}, packet reception ratio and delay.

Parameter	Description	Value
T_l	sleep interval	512 ms
T_s	time of RSS sampling	12 ms
T_t	time of tail	20 ms
T_a	on-air time per preamble packet	[576, 4256] us
T_p	time of preamble	532 ms

Fig. 4: Default settings of Box-MAC LPL in TinyOS.

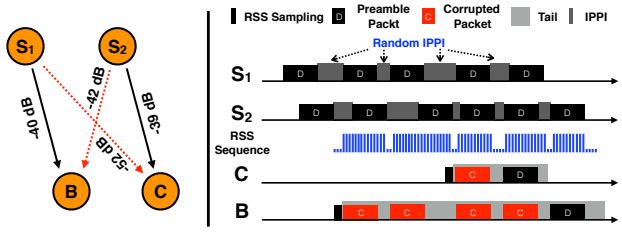


Fig. 5: Overview of *Chase*.

We further examine the impact of packet mishear and collision on delay. We measure the flooding delay of **C** and **B** under clear environment. As shown in Figure 3(c), the broadcast delay is almost uniformly distributed between 0 and T_l in clear environment. However, the delay is significantly increased due to packet mishear and packet collision. About 36.2% of delay is larger than T_l due to packet mishear. Moreover, over 86.2% of delay is larger than T_l due to packet collision.

To conclude, due to the long time transmission of multiple preamble packets and bursty broadcasts, the probability of packet contentions and collisions is high in LPL network flooding, which further increases the flooding delay. Concurrent broadcast in LPL is not well supported by existing strategies. Thus, we need to develop a practical strategy for reliable concurrent broadcast in LPL. With reliable concurrent broadcast, each node can immediately broadcast its received packet during flooding. The speed of network flooding is accelerated.

III. *Chase* Design

The goal of *Chase* is to improve reliability of concurrent broadcast in LPL. To achieve the design goal, there are several requirements:

- First, to support concurrent packet transmission, there are strict requirements on packet transmission. We leverage capture effect in *Chase* by distributedly and constructively satsifying two requirements: 1) signal time: the strongest signal must be received no later than 160μs after the first signal, and 2) signal strength: the strongest signal must be 3 dB larger than the sum of other signals.
- Second, when the signal time or the signal strength requirement cannot be satisfied due to physical constraint, *Chase* should ensure that broadcast packet can also be successfully delivered.

A. Design Overview

We illustrate the design overview of *Chase* in Figure 5. The design of *Chase* mainly consists of two components.

First, instead of using explicit signal time controlling technique as in existing approaches, a randomized IPPI technique is proposed to satisfy the signal time requirement. The technique is fully distributed while introduces no additional overhead. With randomized IPPI technique, we show that as long as the signal strength requirement of multiple received signals can be satisfied, receiver can successfully receive a preamble packet in short time.

Second, it is also possible that signal strength cannot be satisfied. We propose a signal pattern based tail extension method. With such a method, each node can detect whether there are broadcast packet collisions even without receiving packets. If there are, the node will extend the radio on time until a packet is successfully received.

We take Figure 5 as an example to illustrate the principles of *Chase*. $\mathbf{S}_1$ and $\mathbf{S}_2$ concurrently broadcast the flooding packet. The average received RSS difference between strong and weak signals is 2 dB for **B** and 13 dB for **C**. Compared with weak signal from $\mathbf{S}_1$, the time offset for strong signal from $\mathbf{S}_2$ varies with random IPPI. As long as there exists a difference less than 160μs, **C** can receive the $\mathbf{S}_2$'s preamble packet.

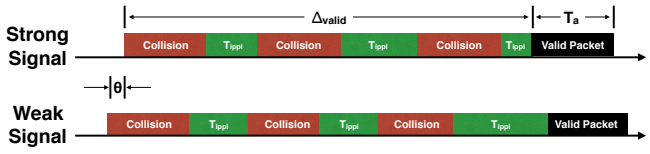


Fig. 6: Definition of valid packet that satisfies the time requirement of capture effect in concurrent broadcast.

Meanwhile, due to signal strength constraint, **B** in no means can successfully receive any overlapped preamble packet as shown in Figure 5. Therefore, **B** will turn off the radio and thus miss the chance to hear the following preamble packets in LPL. In such a case, **B** needs to first detect the existence of collided broadcast packets. *Chase* achieves this by analyzing the signal pattern since the signal pattern of collided broadcast packets is different from that of single LPL transmission (details are in Section III-C). We add a time extension to T_t for node **B** such that **B** can receive the incoming preamble packets after $\mathbf{S}_1$ or $\mathbf{S}_2$ finishes its transmission.

In asynchronous duty cycle network flooding, it is rare that all broadcast packets from different senders (e.g., **S1** and **S2** in Figure 5) are transmitted at the same time. Once it happens, receiver (e.g., **B** in Figure 5) may fail to receive any preamble packet. In such a case, the receiver will immediately send a request to ask senders to rebroadcast when the signals of collided broadcast packets disappear, but no broadcast packet is successfully received. When sender receives the request from receiver, it will start to broadcast after a random backoff.

In Figure 5, we illustrate the situation of two senders. It is possible more than two senders exists. In capture effect, receiver can classify multiple transmissions into two categories, the strongest signal and all others. More specifically, as in other capture effect based protocol designs [7] [13], the sum of all other signals can be treated as a signal. Thus, *Chase* can work in the situation of multiple senders.

B. Random IPPI Adjustment

To reduce packet collision, we set the interval T_{ippi} to be random in order to achieve a time offset to satisfy the time requirement. Meanwhile, T_{ippi} must be less than RSS sampling time T_s to avoid that receiver misses the rendezvous with sender. In *Chase*, we require T_{ippi} be in the range $[0, T_s - \delta]$, where δ serves as a guard time. Assume f_{clock} is the frequency of MCU clock, T_{ippi} is mapped to random integer X in the range of $[0, (T_s - \delta) f_{clock}]$. As shown in Figure 6, the *valid packet* indicates the preamble packet satisfies the time requirement of capture effect. Given the on-air time T_a of preamble packet and the initial time offset θ, the objective is to choose X to minimize the expected delay Δ_{valid} before *valid packet*.

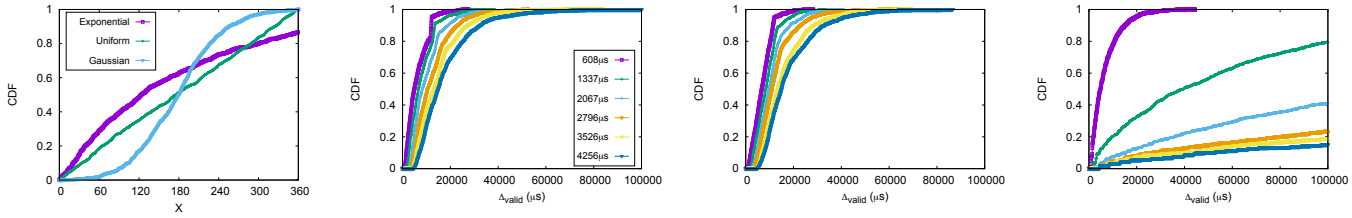
To evaluate the impact of X, we set the distribution of X to be *Exponential*, *Uniform* and *Gaussian* distribution and calculate the expected Δ_{valid}. On TelosB node, the finest granularity of stable MSP430 MCU clock is 32768 Hz. According to default Box-MAC [21], we set T_s and δ to 12ms and 1ms respectively. Thus the range of T_{ippi} is in [0,11]ms. It is discretized to about 360 values. The range of X is [0, 360]. Correspondingly, the expected value of uniform distribution is 180. The λ of exponential distribution and μ of gaussian distribution are 180. For exponential distribution, the range of X is [0, $+\infty$) so that we set X to 360 for $X > 360$. For gaussian distribution, the σ is set as 60 so that most of random values fall into [0, 360]. The CDFs of three distributions of X are shown in Figure 7(a). To show the influence of T_a on X selection, we select 6 different values of T_a ranging from 574μs to 4256μs.

The results are shown in Figure 7. We can see that for all three distributions, Δ_{valid} increases while T_a increases. The reason is that the length of overlapped signals is long with large T_a. The long overlapped signals need more time to reverse their order. Moreover, compared with *Uniform* (Figure 7(c)) and *Exponential* (Figure 7(b)), the Δ_{valid} of *Gaussian* distribution (Figure 7(d)) increases much faster. No more than 20% of Δ_{valid} is less than 100000μs when T_a is larger than 2796μs. The reason is that the variance of T_{ippi} under gaussian distribution is less than the other two distributions as shown in Figure 7(a). It needs more time to reverse the order of overlapped signals with small variance of T_{ippi}. The trend of *Exponential* (Figure 7(b)) and *Uniform* distribution (Figure 7(c)) is much similar. After about 5000μs waiting, it is possible to capture one valid packet when T_a is 4256μs. After 100000μs waiting, at least one valid packet can be captured for all T_a.

With *Exponential* distribution, as shown in Figure 7(a), the probability of selecting large ($X > 280$) and small ($X < 147$) T_{ippi} is higher than *Uniform*. On one hand, the T_{ippi}s of weak and strong signals separately fall into small and large part with high probability. Due to large variance, Δ_{valid} can be short. On the other hand, it is possible that T_{ippi} of weak and strong signals simultaneously falls into the same small or large time interval. Due to the lack of variance, Δ_{valid} is enlarged. In Figure 7(b) and Figure 7(c), we can see the ratio of small Δ_{valid} in *Exponential* distribution is higher than that in *Uniform*. The ratio is getting lower as the increasing of T_a due to long overlapped signals. The maximum Δ_{valid} of *Exponential* distribution is larger than *Uniform*. The difference of the maximum Δ_{valid} between these two distributions increases from about 1ms to 18ms with the increasing of T_a. In summary, *Chase* chooses *Exponential* distribution when T_a is smaller than a threshold T_x and *Uniform* distribution for the rest of T_a.

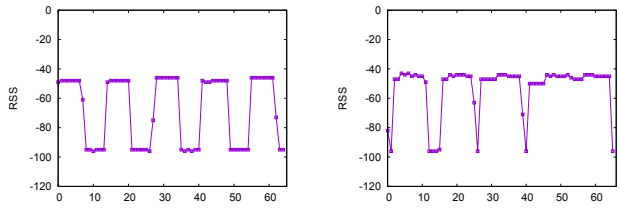
C. Tail Extension Strategy

It is possible that the requirements of capture effect cannot be satisfied in T_t. In such a case, the receiver will extend its tail time in order to receive more preamble packets when collision occurs. Thus our objective is to distinguish the collided broadcast packets. Here, we exploit the RSS features resulted from the sum of preamble packets with random IPPI. More



(a) X distribution (b) Exponential Distribution (c) Uniform Distribution (d) Gaussian Distribution

Fig. 7: (a) shows CDFs (Cumulative Distribution Function) of three distribution of random variable X. The CDF of Δ_{valid} given (b) $X \sim Exponential(1/180)$, (c) $X \sim Uniform(0, 180)$ and (d) $X \sim Gaussian(180, 60)$ separately with diverse $T_a \in [576, 4256]$.



(a) Signal transmission (b) Concurrent broadcast

Fig. 8: Illustration of the difference between RSS sequences of (a) signal transmission and (b) concurrent broadcast.

specifically, we explore two features (i.e., variances of on-air time and segment interval) of continuously sampled RSS sequence to detect the collided broadcast packets. If collided broadcast packets are detected and no preamble packet is successfully received, T_t is extended.

1) RSS Sequence Sampling and Features: After each node wakes up, it continuously samples the RSS. The RSS sampling rate is f_s. The sampled RSS sequence is denoted as $R = \{r_1, r_2, ..., r_n\}$ in tail. Take TelosB node as an example, the f_s is set as about 3 samples per microsecond. When T_t, T_a and T_s is 20ms, 3ms and 4ms, respectively, the sampled RSS sequences of single transmission and concurrent broadcast are shown in Figure 8. The bottom RSS (about -96 dB) indicates the *noise floor*, which corresponds to the signal strength of background noise. The consecutive samples that are higher than noise floor consist signal *segment*. Each RSS sequence consists of segments connected by noise floor.

The first feature is the variance of on-air time, which indicates the time difference of signal segments. As shown in Figure 8(a), the on-air time of each segment of single transmission is a fixed value in the range of $[574, 4256]\mu$s. However, as shown in Figure 8(b), due to overlap of multiple preamble packets of different senders in concurrent broadcast, the on-air time of each segment varies and may be longer than that of single transmission. The on-air time of each segment is randomly distributed in the range of $[574, +\infty)\mu$s. Thus the variance of on-air time of single transmission is almost zero, while that of concurrent transmission is not.

The second feature is variance of segment interval, which indicate the difference of the interval between two adjacent segments. For single transmission, the segment interval, namely T_{ippi}, is almost fixed, as shown in Figure 8(a). However, with the random IPPI adjustment, the segment intervals of concurrent broadcast are randomly distributed in the range of $[0, T_s - \delta]$. Thus, the variance of segment interval of concurrent broadcast is larger than that of single transmission. The detailed steps for extracting features are as follows.

2) RSS Sequence Segmentation: Given RSS sequence $R = \{r_1, r_2, ..., r_n\}$, the objective of segmentation is to extract segments. If the RSS value increases from noise floor (denoted as $Noise$) by a threshold Δ_{rss}, a start point is detected. Similarly, when a RSS sample falls back to noise floor, an end point is detected. Thus, the sets of start (S) and end (E) points of segments are:

$$S = \{s | |r_{s-1} - Noise| < \Delta_{rss}, |r_s - Noise| \geq \Delta_{rss}\} \quad (1)$$

$$E = \{e | |r_{e-1} - Noise| \geq \Delta_{rss}, |r_e - Noise| < \Delta_{rss}\} \quad (2)$$

We set S and E in ascending order and put them in two separated arrays I_S and I_E. We use $I_S(k)$ and $I_E(k)$ to indicate the k^{th} start and end points separatively. It is possible that the start point of the first segment or the end point of the last segment are not in S or E. To address the first case, we remove $I_E(1)$ from I_E if $I_S(1)$ is larger than $I_E(1)$. To address the second case, we remove $I_S(|I_S|)$ from I_S if $I_E(|I_E|)$ is smaller than $I_S(|I_S|)$. Hence, we have $|I_S|$ equals to $|I_E|$ as the total number of segments K. The k^{th} segment can be represented by $R_k = \{r_{I_S(k)}, r_{I_S(k)+1}, ..., r_{I_E(k)}\}$.

3) Feature Extraction: **Variance of On-air Time.** The on-air time of the k^{th} segment is calculated as:

$$T_{on}(k) = (I_E(k) - I_S(k))\frac{1}{f_s} \quad (3)$$

where f_s is RSS sampling rate. For K segments of RSS sequence R, the variance of on-air time is calculated as the difference between the largest and smallest T_{on}.

$$V_{on}(R) = max\{T_{on}(i) - T_{on}(j) | 1 \leq i, j \leq K\}. \quad (4)$$

Variance of segment interval. The segment interval between the k^{th} and $(k+1)^{th}$ segments is calculated as:

$$SegI(k) = (I_S(k+1) - I_E(k))\frac{1}{f_s} \quad (5)$$

where f_s is RSS sampling rate. For K segments of RSS sequence R, the variance of segment interval is the maximum gap among total $K-1$ segment intervals.

$$V_{segi}(R) = max\{SegI(i) - SegI(j) | 1 \leq i, j \leq K-1\} \quad (6)$$

4) Decoded Preamble Packets Mapping: For overlapped preamble packets, one segment at least contains one preamble packet. For RSS sequence R, decoded vector D are used to indicate whether the signals of K segments are successfully decoded i.e.,

$$D_i = \begin{cases} 0, & \text{No decoded preamble packet} \\ 1, & \text{Otherwise} \end{cases} \quad (7)$$

for $1 \leq i \leq K$. If all elements in D are zero, all preamble packets in T_t are corrupted. Then, receiver needs further to extend T_t when these corrupted preamble packets belong to concurrent broadcast. Otherwise, receiver will start to broadcast when the decoded preamble packet is a flooding packet.

5) Identification Algorithm: Given the feature tuple (V_{on_air}, V_{segi}) of sampled RSS sequence R and decoded vector D, Algorithm 1 shows the process to verify whether collided broadcast packets exist. At line 1, if one preamble packet is successfully received, receiver needs not extend T_t. At line 3, κ is the threshold of variance of on-air time. τ is the threshold of variance of segment interval. As shown in Section III-C1, when there is no concurrent transmission, κ and τ is close to 0. Otherwise, κ and τ is usually larger than 0, due to the diverse pattern of signal overlapping. T_t will not be extended if both V_{on} and V_{segi} of the signal are smaller than κ and τ. Thus, with smaller κ and τ, the signal tends to be identified as concurrent transmission. The time and space complexity of identification process is $O(K)$.

Algorithm 1 Identification Algorithm

Input: Feature temple (V_{on}, V_{segi}), decoded vector D.
Output: Whether the corrupted preamble packets belong to concurrent broadcast.

1: **if** $\exists i \in [1, K]$, D_i equals to 1. **then**
2: return FALSE.
3: **else if** $V_{on} < \kappa$ and $V_{segi} < \tau$. **then**
4: return FALSE.
5: **end if**
6: return TRUE.

D. Influence on Other Traffics

It is possible that both flooding and other network traffics coexist in networks. *Chase* can also work with other kinds of network traffics, for instance, for widely used data collection traffic [1] [9], where packets are forwarded to sink with multi-hop unicast relay. In LPL, unicast traffic adopts carrier sense and random backoff to avoid packet collision. With *Chase* broadcast, unicast traffic can be transmitted when the channel is temporally clear. Thus the efficiency of those kinds of traffic will not be influenced by *Chase*.

TABLE I: The summarization of system parameter settings

Parameter	Description	Value
T_x	T_a boundary of random function	2067μs
δ	guard time of T_{ippi}	0.1ms
f_{clock}	stable MCU clock	32768Hz
f_s	RSS sampling rate	31250Hz
Δ_{rss}	RSS threshold of segmentation	3 dB
κ	threshold of on-air time variance	64μs
τ	threshold of segment interval variance	64μs

IV. Implementation

We implement *Chase* with TinyOS 2.1.2 on TelosB nodes. Besides the default Box-MAC LPL parameters as shown in Figure 4, other parameter settings are summarized in Table I.

A. Random Function

We empirically choose the boundary T_x as 2067μs. The decision criteria of X distribution is shown in Equation 8.

$$X \sim \begin{cases} Exp(2f_{clock}/(T_s - \delta)), & 0 < T_a <= 2067 \\ Uni(0, f_{clock}(T_s - \delta)), & 2067 < T_a <= 4256 \end{cases} \quad (8)$$

To ensure the stability, we select the clock source of f_{clock} as the watch crystal of MSP430f1612 MCU with frequency 32768 Hz on TelosB. The guard time δ is set as 0.1ms to make sure T_{ippi} is smaller than T_s and keep the range of T_{ippi} large.

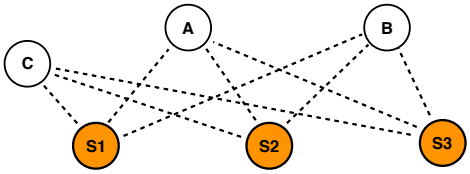
We use LCG (Linear Congruential Generator) to obtain uniform distribution. To guarantee the diversity among different nodes, the initial seed is set according to node ID. Further, we generate exponential distribution by the transformation of uniform distribution.

B. Precise IPPI Control

It is important to guarantee the actual T_{ippi} is exactly corresponding to the random function. MCU controls the operation of radio through SPI. Due to the arbitration of SPI resource in TinOS, MCU can not transmit any packet when it uses SPI to read the data of received packet from radio buffer (RxFIFO in CC2420). To remove the potential delay of resource arbitration, sender disables the interrupt service of packet reception during broadcast. For CC2420 radio on TelosB, the packet reception can be turned off by strobing SRFOFF register.

C. RSS Sampling Control

After TelosB node has detected wireless signals, it continuously samples RSS by reading the register RSSI.RSSI_VAL of CC2420. The higher the frequency of MCU DCO (Digital Crystal Oscillator) is, the faster the RSS sampling rate is. The maximum frequency of MCU DCO is about 4MHz. To achieve higher RSS sampling rate, we set MCU DCO frequency as 4MHz after radio has been turned on. The resulted RSS sampling rate f_s is 31250Hz (i.e., 32μs per sampling) to obtain fine grain channel profile. To ensure the detection reliability of collided broadcast packets, we empirically set the threshold of κ and τ as a small value 64μs, which is the time of two RSS samples. In 20ms tail, the time of RSS sampling is set as



RSS(dBm)	A	B	C
S1	-45	-58	-34
S2	-44	-51	-44
S3	-48	-53	-64

Fig. 9: Illustration of the topology and link state in controlled experiments.

16ms. Total 500 RSS samples can be recorded. The average time of RSS sequence processing is about 2.73ms, which is completely covered by the 4ms rest tail. If no preamble packet is successfully received and collided broadcast packets are detected in tail, the tail is extended another 20ms each time.

Moreover, due to the SPI resource arbitration, the RSS.RSSI_VAL register can not be accessed when MCU reads the received data from radio buffer (RxFIFO in CC2420). Even with 4MHz MCU DCO, RxFIFO buffer swapping takes about [130, 1280]μs for different T_a. The long time blank space of RSS sampling incurs uncertainty on segmentation. In our implementation, instead of reading all data of RxFIFO, we directly read the CRC byte after the frame length byte has been read. If the packet CRC is valid, we read the rest of bytes. Otherwise, we flush RxFIFO and continuously sample RSS again. The delay of reading length and CRC bytes is only about 20 μs, in which few RSS sample is lost.

V. Evaluation

In this section, we verify the efficiency of *Chase* through both controlled and testbed experiments.

A. Random Strategy Efficiency

We use 6 TelosB nodes in control experiments. The topology and link state are shown in Figure 9. **A**, **B** and **C** are three receivers, waiting for packets broadcasted by three senders **S1**, **S2** and **S3**. The power is set to 7 and the average receivers' RSS of the packets from different senders is shown in Figure 9.

1) Delivery Reliability: For each testing round, three senders concurrently broadcast 100 packets with different sequence number to receivers. To guarantee concurrent broadcast of each packet, all senders are synchronized at the initialization phase and start to broadcast preamble packets with a random [5,100]ms delay to imitate the asynchronous transmission in practical. At the end of each testing round, each receiver calculates the packet reception ratio (PRR), i.e., the number of received non-duplicate packets divided by 100. For each experiment setting, we run 30 testing rounds and calculate the average PRR and radio duty cycle of individual receiver to measure the delivery reliability and energy efficiency. The packet length is 37 bytes, with an on-air time of about 1.34ms. We shorten T_s to 2.9ms for reducing the baseline of energy consumption as Zisense[28] does.

To examine the influence of different components of *Chase* on delivery reliability, we compare *Chase* with other three concurrent transmission strategies: (1) set IPPI as zero (default LPL), (2) set a large and fixed IPPI as 2ms (adaptive LPL for concurrent broadcast [17]), and (3) use uniform random IPPI in the range of [0,2.8]ms (optimized LPL for concurrent broadcast in Section III-B).

The results are shown in Figure 10. With large IPPI 2ms, the PRR of all receivers is increased, about 2.7 times improvement for **A**. The reason is that 2ms IPPI leaves more space for those preamble packets transmitted when other senders are waiting during IPPI. Moreover, the PPR of all receivers further increases and becomes larger than 85% with random IPPI. Compared with fixed IPPI, random IPPI makes different arrival timing for different overlapped preamble packets to avoid the strongest signal arrives too late every time. With large and random IPPI, the increasing of duty cycle also indicates the receivers have more chances to successfully receive packet than zero IPPI.

Further, the PRR of **A** and **B** is increased to nearly 100% with *Chase*. As shown in Figure 9, at **A** and **B**, the RSS difference between strong and weak signals may be smaller than 3 dB so that capture effect does not work. Without capture effect, the fixed tail 20ms may not be long enough to resolve continuous collision of preamble packets. In *Chase*, the adaptive tail extension further ensures the delivery reliability.

In above experiments, the delivery reliability of *Chase* achieves almost 100% under fixed T_a (1.34ms) and T_s (2.9ms). We further test the delivery reliability under different T_a and T_s on the controlled topology. The results are shown in Figure 11. When T_a is 1344μs, the average PRR is close to 100% under all ZigBee detection time T_s. However, as increasing of T_a, the PRR falls down with the decreasing of T_s. The reason is that when T_s is small, the range of random T_{ippi} becomes small. With large T_a of preamble packet, the frequency of long overlapped ZigBee signals increases so that the number of segments in RSS sequence may decrease. With few segments, false negative ratio of identification algorithm increases. Therefore, receiver fails to extend the tail. Thus the loss of broadcast packet occurs. However, the probability of false negative is relatively low, the average PRR is still higher than 95% in the worst case.

2) Energy Efficiency: With the controlled topology, we test the distribution of the length of tail to successfully receive one preamble packet under different T_a and T_s. The results are shown in Figure 12. When $T_s > 7$ms, for all different T_a, the average tail length is about 20ms and the maximum tail length is no larger than 60ms. However, when T_s is less than 7ms, the tail length with long T_a increases faster than it with short T_a. The reason is that short random range is probably not long enough to construct capture effect, especially for long T_a. It needs to extend the tail to wait for the receiving opportunity of non-collision preamble packets. When T_a increases to 3904μs, the average tail length increases to about 270ms and the maximum tail length is 440ms, about 13.6 and 22 time of

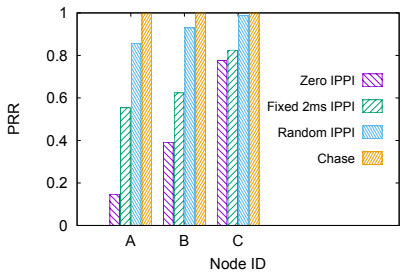


Fig. 10: PRR performance under different IPPI adjustment strategies.

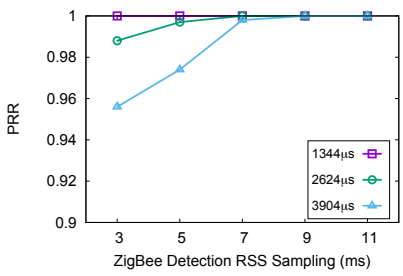


Fig. 11: PRR performance under different detection RSS sampling duration.

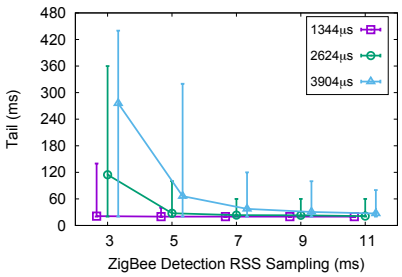


Fig. 12: Tail length under different detection RSS sampling duration.

TABLE II: The Accuracy of Identification Algorithm

	Correct	False Negative	False Positive
Concurrent Broadcast	98.7%	1.3%	0
Single Transmission	100%	0	0
Channel Contention	99.2%	0	0.8%
Hidden Terminal	30.5%	0	69.5%

default tail length. Thus, when T_a is long, it is better to set T_s larger than 7ms to keep energy efficiency in *Chase*,

To conclude, *Chase* largely improves the delivery reliability of concurrent broadcast under various settings. The cost of reliable concurrent broadcast becomes larger when the gap between T_a and T_s becomes smaller.

B. Identification Accuracy

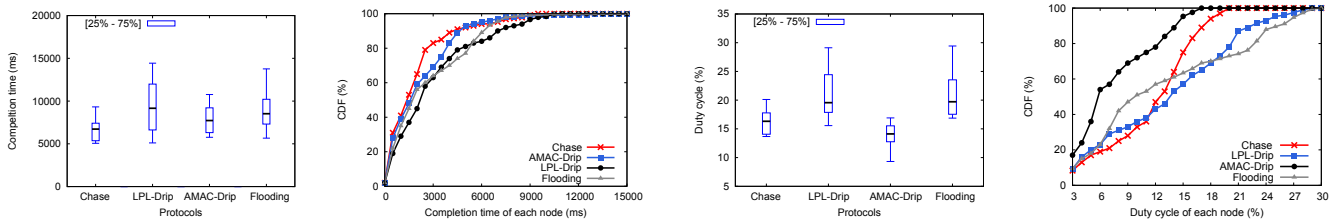
According to the performance analysis of delivery reliability in the controlled experiments, the accuracy of identification algorithm is key factor. With the controlled topology shown in Figure 9, we make **A**, **B** and **C** identify whether the type of received signals is concurrent broadcast with *Chase*. We test the identification accuracy for 4 types of data flows. The first type is all three senders concurrently broadcast. The second type is only one sender unicasts or broadcasts. The third type is all three senders content the channel to unicast or broadcast. The last type is all three senders are hidden terminal and concurrently unicast or broadcast. For each transmission of different data flows, the T_a and T_s is randomly chosen. Each node identifies 1000 RSS sequences for each type of data flow.

The results are shown in Table II. 98.7% of concurrent broadcast can be correctly identified. The rest of 1.3% is false negative due to the lack of features when the number of segments is small since the signal overlapping is severe. For single transmission and channel contention, the correctness of identification is very high, i.e., 100% and 99.2%. With similar random segment interval and on-air time in channel contention, the false positive may occur when no preamble packet is successfully received. However, the loss of preamble packet is rare due to contention backoff, the false positive is only 0.8%. For hidden terminal, only 30.5% can be correctly identified. The reason is that IPPI and on-air time are much similar between hidden terminal and concurrent broadcast, it fails to identify whether corrupted preamble packets belong to concurrent broadcast. To conclude, *Chase* can correctly recognize concurrent broadcast in most cases. Thus, the delivery reliability of *Chase* is guranteed. For other data flows, the false tail extension may appear when hidden terminal is severe.

Fig. 13: The picture of real testbed with 50 TelosB nodes.

C. Network Flooding

We evaluate *Chase* on the real testbed with 50 TelosB nodes, which are deployed as a grid topology as shown in Figure 13. The distance between two adjacent nodes of both vertical and horizontal direction is about 20cm. We set the radio output power to 3. With such a power, the hidden terminals and packet contentions often appear in the centre area during flooding. The packet length is 77 bytes corresponding to on-air time 2623μs. Another parameters follow the settings in Section IV. We also control the reception range of each node to ensure the enough multi-hop transmission on our testbed. The network density is 4 neighbors per node on average. The network diameter is about 10 hops. We compare *Chase* with three other protocols. The first is deluge flooding with Box-MAC, in which node will immediately broadcast the received packet with duplicate suppression [12] and carrier sense. The second is the widely used flooding protocol Drip [25] with Box-MAC. In Drip, the broadcast is controlled by trickle timer [15] after receiving the packet to further reduce the influence of collision. The last is Drip with AMAC [6], which is the state-of-the-



(a) Distribution of network completion time in different protocols

(b) CDF of per node completion time in different protocols

(c) Distribution of average duty cycle in different protocols

(d) CDF of per node duty cycle in different protocols

Fig. 14: The illustration of the results for real testbed experiments. (a) and (c) show the distribution of the network completion time and average duty cycle of all flooding packets in different protocols. (b) and (d) show the CDF of the per node completion time and duty cycle of all flooding packets in different protocols.

art energy efficient receiver-initiated asynchronous duty cycle MAC.

We show the efficiency of *Chase* through testbed experiments in terms of the completion time and the energy consumption measured by radio duty cycle. For each protocol, the sink continuously floods 100 packets to collect performance data. With *Chase*, multiple senders can concurrently broadcast with reliable packet delivery. The idle radio waiting and packet retransmission can be significantly reduced. Thus, both completion time and energy consumption of *Chase* are more efficient than other flooding protocols in asynchronous duty cycle networks. The results are shown in Figure 14.

As shown in Figure 14(a), the network completion time of *Chase* is smaller than the others three protocols. The median network completion time of *Chase* is about 6723ms. The median network completion time of deluge flooding is 8523ms and LPL-Drip is 9157ms. The improvement is 21.1% and 26.6% with *Chase*. The network completion time of AMAC-Drip is smaller than LPL-Drip due to more efficient channel access. Compared with the AMAC-Drip whose median network completion time is about 7723ms, the improvement of *Chase* is 12.9%. The reason is the probe collision of AMAC reduces the reliability and increases the completion time. The completion time of LPL-Drip is more dynamic due to the exponentially increased interval between adjacent broadcast so that severe packet loss incurs more delay on the interval.

Figure 14(b) shows the CDF of per-node completion time in different protocols. The improvement of completion time with *Chase* is also observed, since the curve of *Chase* is increasing faster than other three protocols. With *Chase*, 81.2% nodes can receive flooding packet within 3000ms. The ratio is only 63.7% and 60.2% with AMAC-Drip and LPL-Drip, respectively.

As shown in Figure 14(c), the average radio duty cycle of *Chase* is smaller than LPL-Drip and deluge Flooding. The median average radio duty cycle of *Chase* is about 16.32%. The median average duty cycle of deluge flooding is 19.72% and LPL-Drip is 19.56%, leading to an improvement of 17.2% and 16.6% by *Chase*. With faster network completion time, *Chase* further provides higher energy efficiency than LPL-based flooding. We observe that the average radio duty cycle of AMAC-Drip is better than *Chase*. Compared with the AMAC-Drip whose median average radio duty cycle is about 14.12%, the degradation is about 13.5%. The reason is the long preamble and extra extension of tail makes high average radio duty cycle of *Chase*, but the idle waiting and tail of AMAC [6] is much smaller. However, as the network completion time of *Chase* is faster than AMAC-Drip, the overall energy consumption of *Chase* and AMAC-Drip is fair.

Figure 14(d) shows the CDF of per node duty cycle in different protocols. The improvement of average radio duty cycle with *Chase* is also observed. We notice that the baseline energy of *Chase* is higher than other three protocols. The reason are *Chase* extends the tail to guarantee the reliability and all nodes will broadcast to relay flooding packet in *Chase*. *Chase* makes the tradeoff between reliability and energy. Considering the faster network completion time, the overall energy efficiency is higher than other flooding protocols.

VI. Related Work

Always-on Radio. Many approaches focus on full-coverage dissemination problem in always-on radio mode. Drip [25] and Deluge [12] are structureless with pure broadcast transmission hop by hop. They utilize the trickle timer [15] to control the dissemination flow for reducing the contention and transmission. ECD [4] further considers the influence of link quality on sender selection. CFlood [30] considers the influence of link correlation on sender selection. Cord [11] and Sprinkler [22] are structure based. An approximate minimum dominating set of nodes are selected as core nodes. *Chase* is based on LPL asynchronous duty cycle radio mode and enables reliable concurrent broadcast to accelerate flooding.

Synchronous Duty Cycle Radio. Synchornization is required in synchronous duty cycle protocols. Glossy [7] exploits the *constructive interference* to fast flooding the data in network wide. Splash [3] adopts the reverse data forwarding structure to increase reliability. Pando [5] further explores the fountain code to reduce the number of retransmission. With local synchronization, each node knows the sleep schedule of its neighbors. The sender just begins transmission after the receiver turns on the radio. Based on the energy-optimal tree, S. Guo et al. [10] exploit opportunistic chance over unreliable

links, which can reduce the expected end-to-end delay, as relay. In contrast, *Chase* works in an asynchronous way. Both synchronous and asynchronous duty cycle models are widely adopted and used in different scenarios. *Chase* is proposed to mainly address the inefficiency of flooding in asynchronous duty cycle protocols.

Asynchronous Duty Cycle Radio. With receiver-initiated asynchronous duty cycle radio mode, the broadcaster transmits the packet to the receivers after successfully receives the receivers' probes. ADB [23] utilizes the progress information of local neighbors to select broadcaster to increase the delivery reliability and reduce the energy consumption. *Chase* enables reliable concurrent broadcast in the sender-initiated asynchronous duty cycle flooding.

VII. Conclusion

We present *Chase*, an efficient and fully distributed concurrent broadcast layer for flooding in asynchronous duty cycle networks. In *Chase*, we propose a distributed random inter preamble packet interval adjustment approach to meet the strict time and signal strength requirements for concurrent transmission. In case that the time and signal requirement cannot be satisfied (e.g., the difference of signal strength is less than a 3 dB) due to physical constraint, we propose a light-weight signal pattern recognition based approach to identify such a circumstance and extend radio-on time to resolve the collision. We implement *Chase* in TinyOS and TelosB node. The implementation can be used as a building block for upper layer protocols. The evaluation results show the effectiveness of *Chase* in asynchronous duty cycle networks.

Acknowledgment

We gratefully acknowledge our shepherd Ting Zhu and ICNP reviewers. Jiliang Wang is the corresponding author. This study is supported in part by the NSFC program under Grant 61472217, the NSFC program under Grant 61572277, the NSFC key program under Grant 61532012, the NSFC Joint Research Fund for Overseas Chinese Scholars and Scholars in Hong Kong and Macao under grant 61529202, and the NSFC program under Grant 61672320.

References

[1] Z. Cao, Y. He, and Y. Liu. L^2: Lazy forwarding in low duty cycle wireless sensor networks. In *Proceedings of INFOCOM*, 2012.

[2] M. H. Daibo Liu, Zhichao Cao and Y. Zhang. Preamble counter: Achieving accurate and real-time link estimation in low power wireless sensor networks. In *Proceedings of IPSN*, 2016.

[3] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: fast data dissemination with constructive interference in wireless sensor networks. In *Proceedings of NSDI*, 2013.

[4] W. Dong, Y. Liu, C. Wang, X. Liu, C. Chen, and J. Bu. Link quality aware code dissemination in wireless sensor networks. In *Proceedings of ICNP*, 2011.

[5] W. Du, J. C. Liando, H. Zhang, and M. Li. When pipelines meet fountain: Fast data dissemination in wireless sensor networks. In *Proceedings of Sensys*, 2015.

[6] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In *Proceedings of Sensys*, 2010.

[7] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of IPSN*, 2011.

[8] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proceedings of OSDI*, 2008.

[9] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of Sensys*, 2009.

[10] S. Guo, L. He, Y. Gu, B. Jiang, and T. He. Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links. *IEEE Transactions on Computers*, 63(11):2787–2802, 2014.

[11] L. Huang and S. Setia. Cord: Energy-efficient reliable bulk data dissemination in sensor networks. In *Proceedings of INFOCOM*, 2008.

[12] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of Sensys*, 2004.

[13] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proceedings of Sensys*, 2013.

[14] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson. Low power, low delay: opportunistic routing meets duty cycling. In *Proceedings of IPSN*, 2012.

[15] P. A. Levis, N. Patel, D. Culler, and S. Shenker. *Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks*. Computer Science Division, University of California, 2003.

[16] D. Liu, Z. Cao, X. Wu, Y. He, X. Ji, and M. Hou. Tele adjusting: Using path coding and opportunistic forwarding for remote control in wsns. In *Proceedings of ICDCS*, 2015.

[17] J. Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *Proceedings of INFOCOM*. 2009.

[18] X. Mao, X. Miao, Y. He, X.-Y. Li, and Y. Liu. Citysee: Urban CO_2 monitoring with sensors. In *Proceedings of INFOCOM*, 2012.

[19] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of Sensys*, 2004.

[20] L. Mo, Y. He, Y. Liu, J. Zhao, S.-J. Tang, X.-Y. Li, and G. Dai. Canopy closure estimates with greenorbs: sustainable sensing in the forest. In *Proceedings of Sensys*, 2009.

[21] D. Moss and P. Levis. Box-macs: Exploiting physical and link layer boundaries in low-power networking. *Technical Report SING-08-00, Stanford*, 2008.

[22] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices. *IEEE TMC*, 2007.

[23] Y. Sun, O. Gurewitz, S. Du, L. Tang, and D. B. Johnson. Adb: an efficient multihop broadcast protocol based on asynchronous duty-cycling in wireless sensor networks. In *Proceedings of Sensys*, 2009.

[24] TelosB. Crossbow inc, 2013.

[25] G. Tolle and D. E. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of EWSN*, 2005.

[26] J. Wang, Z. Cao, X. Mao, and Y. Liu. Sleep in the dins: Insomnia therapy for duty-cycled sensor networks. In *Proceedings of INFOCOM*, 2014.

[27] T. Xiang, Z. Chi, F. Li, J. Luo, L. Tang, L. Zhao, and Y. Yang. Powering indoor sensing with airflows: a trinity of energy harvesting, synchronous duty-cycling, and sensing. In *Proceedings of Sensys*, 2013.

[28] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu. Zisense: towards interference resilient duty cycling in wireless sensor networks. In *Proceedings of Sensys*, 2014.

[29] X. Zheng, J. Wang, W. Dong, Y. He, and Y. Liu. Bulk data dissemination in wireless sensor networks: Analysis, implications and improvement. *IEEE Transactions on Computers*, 65(5):1428–1439, 2016.

[30] T. Zhu, Z. Zhong, T. He, and Z.-L. Zhang. Exploring link correlation for efficient flooding in wireless sensor networks. In *Proceedings of NSDI*, 2010.