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

Zhichao Cao, *Member, IEEE, ACM*, Daibo Liu, *Student Member, IEEE*, Jiliang Wang, *Member, IEEE, ACM*, and Xiaolong Zheng, *Member, IEEE*

***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 time and strength requirements (e.g., Capture Effect) for concurrent broadcast 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 lightweight signal pattern recognition-based approach to identify such a circumstance and extend radio-on time for packet delivery. We implement *Chase* in TinyOS with TelosB nodes 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.**

***Index Terms*— Wireless communication, concurrent broadcast, Internet of Things, network flooding, asynchronous duty cycle.**

I. Introduction

INTERNET of Things (IoT) [23], [25], [33] 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 [12]. Asynchronous duty cycle, namely LPL (Low Power Listening) (e.g., Box-MAC [26], Zisense [34], [35]), is one of the most commonly used low duty cycle modes [2], [19]. 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 radios (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 [4], [6], [27], [36], time stamp [11], [24], etc) to all nodes. Flooding is also widely used to notify nodes [21] and update system parameters [32], 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. Many flooding protocols [5], [15], [16], [27], [31], [37] assume the radio is always on. Based on either structure or structureless methods, they select the optimal set of senders to quickly cover all nodes with the minimum packet transmissions. However, they are hard to adapt the temporal diversity of asynchronous duty cycle in LPL. Combining with explicit wake-up schedules of neighbors and data collection tree structure, Guo *et al.* [14] propose opportunistic flooding for duty cycle network. One practical limitation is that it is hard to obtain the wake-up schedules of all neighbors in dense network. Thus, some delivery chances may be overlooked. Moreover, the wake-up schedules of neighbors and data collection tree structure also need extra maintenance overhead. Further, synchronous flooding protocols (e.g., Glossy [11], Splash [4] and Pando [6]) use constructive interference to fast flood packet after all nodes synchronously wake up. Synchronous flooding protocols fit those applications with periodical flooding demands so that the applications can periodically make all nodes wake up from asynchronous duty cycle. However, they do not work for irregular flooding requests. As a result, network protocols and services (e.g., network parameter update, network notification)

Manuscript received September 8, 2016; revised March 31, 2017; accepted May 23, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor X. Zhou. Date of publication June 28, 2017; date of current version October 13, 2017. This work was supported in part by the NSFC Program under Grant 61472217, Grant 61572277, and Grant 61672320, in part by the NSFC Key Program under Grant 61532012, in part by the NSFC Joint Research Fund for Overseas Chinese Scholars and Scholars in Hong Kong and Macao under Grant 61529202, and in part by the China Postdoctoral Science Foundation under Grant 2016M601034. *(Corresponding author: Daibo Liu.)*

Z. Cao, J. Wang, and X. Zheng are with the School of Software, Tsinghua University, Beijing 100084, China (e-mail: caozc@greenorbs.com; wang@greenorbs.com; xiaolong@greenorbs.com).

D. Liu is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: dbliu.sky@gamil.com).

Digital Object Identifier 10.1109/TNET.2017.2712671

built on top of flooding are also not efficient in LPL network.

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 broadcast has strict requirement on time and signal strength. For example, for capture effect [10], [18] which enables concurrent broadcast, 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 use a random *Inter Preamble Packet Interval* (IPPI) adjustment model to minimize the waiting time to meet a preamble packet that satisfy the time offset requirement of capture effect. We use several common random functions to show that the time offset of capture effect can be achieved with a high probability. Moreover, when the requirements of capture effect cannot be satisfied due to physical constraints, we notice that the shape of the sequence of received signal strength is much different between concurrent broadcast and normal transmission. *Chase* leverages a light-weight classifier of signal pattern recognition to identify concurrent broadcast. When concurrent broadcast is detected and no flood packet is received, Chase extends 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 duty cycle networks. Our contributions are summarized as follows.

- We propose *Chase*, an efficient and fully distributed broadcast layer to support concurrent broadcast for flooding in asynchronous duty cycle networks.
- We address the difficulties in supporting concurrent broadcast in practical asynchronous duty cycle networks and design light-weight and efficient countermeasures.
- We implement *Chase* in TinyOS with TelosB nodes [30]. The evaluation results show that *Chase* can significantly improve flooding speed.

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 details and evaluation results, respectively. Section VI introduces the related work. We discuss several technical issues in Section VII. Finally, Section VIII 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,

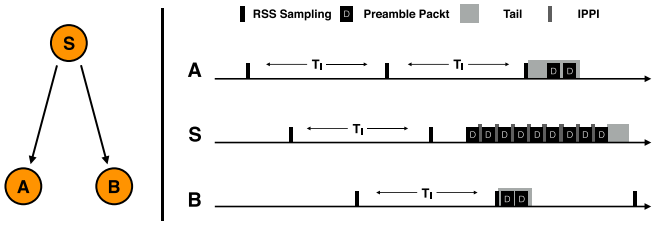


Fig. 1. Illustration of LPL broadcast from sender **S** to its neighbors **A** and **B**.

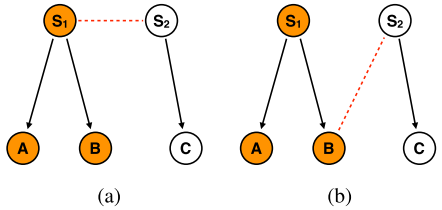


Fig. 2. Illustration of the situation of packet mishear and packet collision. (a) Packet mishear. (b) Packet collision.

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.

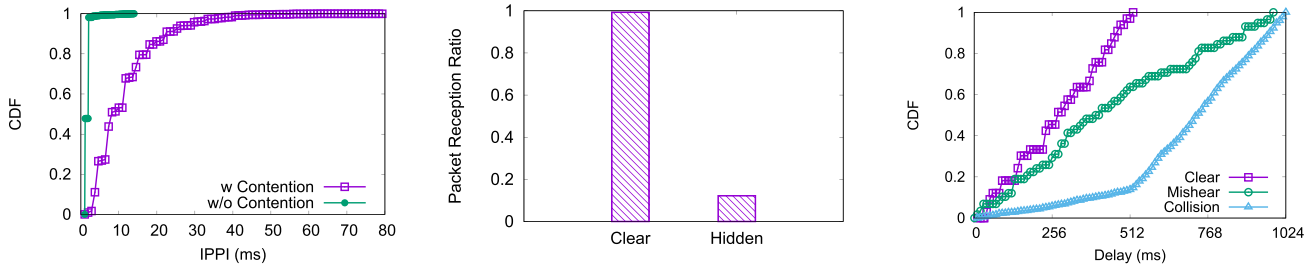
There are two requirements for broadcast in LPL. First, $T_p > T_l$ to ensure **S** can meet **A** and **B** at their rendezvous. Thus, broadcast in LPL will occupy the channel for a long time. Second, $T_s > T_{ippi}$ to ensure **A** and **B** can detect the signal from **S**. In practice, T_t is usually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



(a) (b) (c)

Fig. 3. The impact of channel contention and hidden terminal on T_{ippi}, packet reception ratio and delay. (a) T_{ippi} distribution. (b) Packet Reception Ratio. (c) Delay distribution.

Parameter	Description	Value
T_I	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.

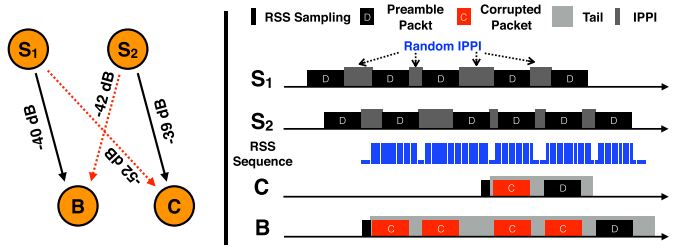


Fig. 5. Overview of *Chase*.

As shown in Figure 4, we use the default settings of Box-MAC [26] 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 coexistent interference (e.g. WiFi, Bluetooth).

First, we use two TelosB nodes $\mathbf{S}_1$ and $\mathbf{S}_2$ as shown in Figure 2(b). 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%.

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 broadcast, 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 (details are in Section III-B). As long as there exists a difference less than 160μs, **C** can receive the $\mathbf{S}_2$'s preamble packet.

Meanwhile, due to signal strength constraint, **B** in no means can successfully receive any overlapped preamble packet as shown in Figure 5. 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.

B. Random IPPI Adjustment

To reduce packet collision, each concurrent sender distributedly and randomly sets its T_{ippi} in order to achieve a time offset to satisfy the temporal constraint of capture effect at a receiver. We first give the basic two-senders model in Section III-B1. Further, we construct multiple-senders model and show the efficiency of random IPPI with increasing of concurrent senders in Section III-B2.

1) Two-Senders Model: In two-senders model, as shown in Figure 6, the sender that provides high RSSI is called strong signal, the other is called weak signal. After a receiver wakes up, it continuously hears some overlapped preamble packets. θ indicates the time offset between the first pair of overlapped preamble packets. If θ does not satisfy the temporal constraint of capture effect, the receiver cannot successfully

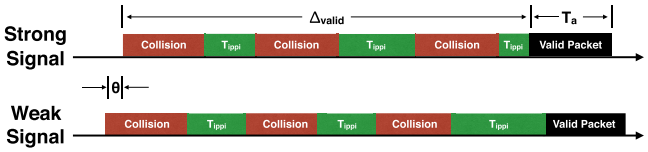


Fig. 6. Definition of valid packet that satisfies the time requirement of capture effect in concurrent broadcast.

decode the preamble packet of strong signal. The question is *how to make a preamble packet of strong signal to fulfill the temporal constraint of capture effect as soon as possible by randomly adjusting* T_{ippi}.

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 $[0, (T_s - \delta) f_{clock}]$. We define *valid packet* to indicate the preamble packet of strong signal that satisfies the temporal constraint of capture effect. In contrast, the other preamble packets are called *collision packet* Given the packet on-air time T_a and the initial time offset θ, our objective is to choose X to minimize the expected delay Δ_{valid} (denoted as $E(\Delta_{valid})$) which is from the beginning of the first collision packet to the coming of the earliest *valid packet*.

We assume that k preamble packets of strong signal and k' preamble packets of weak signal are transmitted during Δ_{valid}, respectively. In Figure 6, both k and k' are 3. The $(k+1)^{th}$ (e.g., 4 in Figure 6) preamble packet of strong signal is the earliest valid packet. We use T_{ippi} and T'_{ippi} to denote the random IPPI for strong signal and weak signal, respectively. T^i_{ippi} and T'^i_{ippi} denote the i^{th} IPPI between the i^{th} and $(i+1)^{th}$ preamble packets. X_i and X'_i denote the corresponding random integers of T^i_{ippi} and T'^i_{ippi}. We have

$$
\begin{aligned}
&\textbf{min } E(\Delta_{valid}) \\
&\quad \text{where } \Delta_{valid} = kT_a + \sum_{i=1}^{k} T^i_{ippi} \\
&\textbf{s.t. } T_a \in [574, 4256]\mu s && 1 \\
&\quad \theta \sim \text{uniform}((160, T_a]) && 2 \\
&\quad (k'-k)T_a + \sum_{j=1}^{k'} T'^j_{ippi} - \theta \\
&\quad - \sum_{i=1}^{k} T^i_{ippi} + 160 \geq 0 && 3 \\
&\quad (k-k')T_a + \sum_{i=1}^{k} T^i_{ippi} \\
&\quad - \sum_{j=1}^{k'-1} T'^j_{ippi} + \theta \geq 160 && 4 \\
&\quad \forall i \in [1,k], T^i_{ippi} = X_i / f_{clock} && 5 \\
&\quad \forall j \in [1,k'], T'^j_{ippi} = X'_j / f_{clock} && 6 \\
&\quad X, X' \sim f(X) && 7 \\
&\quad \int_0^{(T_s-\delta) f_{clock}} f(X) dX = 1 && 8
\end{aligned} \tag{1}
$$

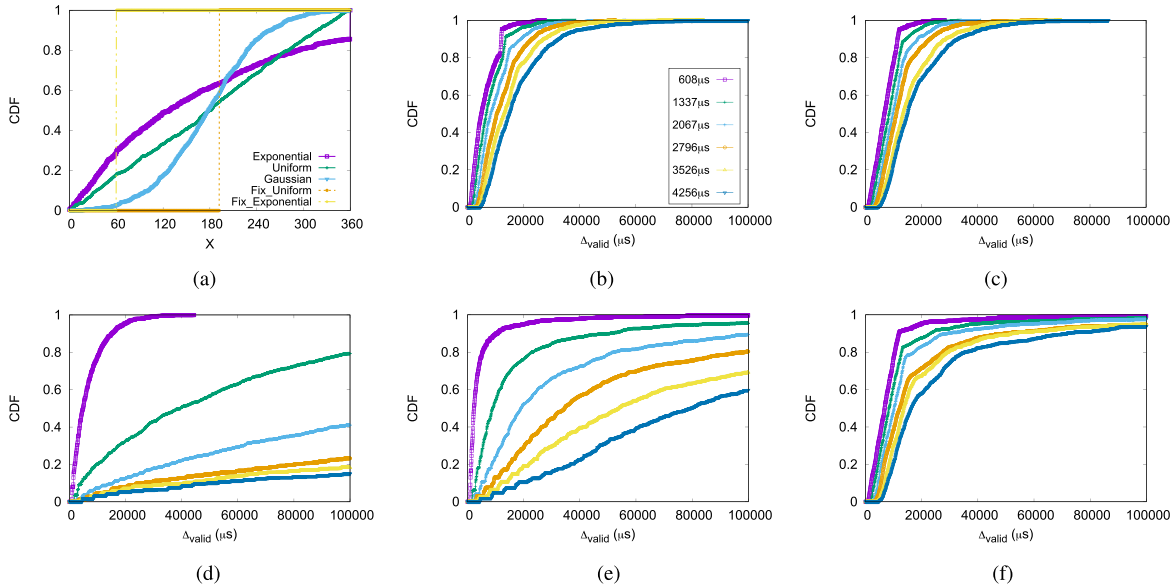


Fig. 7. (a) shows CDFs (Cumulative Distribution Function) of three distribution of random variable X. The CDF of Δ_{valid} given (b) $X \sim Exponential(1/180)$, (c) $X \sim Uniform(0, 180)$ and (d) $X \sim Gaussian(180, 60)$ separately with diverse $T_a \in [576, 4256]$. (e) Fixed Exponential Distribution.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

The constraint 1 defines the range of T_a. The constraint 2 defines the random initial time offset θ corresponds to an uniform distribtuion. The constraint 3 means that the valid packet of strong signal comes no later than 160μs after that of weak signal. This corresponds to temporal constraint of capture effect. The constraint 4 indicates that the valid packet of strong signal comes a preamble length (i.e., 160μs) later than the end of the last collision packet of weak signal. This is a sufficient condition to ensure that no collision packet of weak signal already triggers receiving process when the preamble of valid packet is coming. The constraints 5 and 6 show the relationship between each IPPI of strong and weak signals and the random integers X and X'. Due to the feasibility of system deployment and management, all nodes usually take the same random strategy. The constraint 7 means X and X' are independent and identically distributed. $f(X)$ indicates the probability density function. The constraint 8 shows the range of X is $[0, (T_s - \delta)f_{clock}]$.

To evaluate the impact of X, we use two common strategies. First, X varies for each IPPI. We set three distributions of X, namely *Exponential*, *Uniform* and *Gaussian*. X randomly varies and follows the corresponding probability distribution (i.e., exponential, uniform, gaussian). Second, X is fixed, but its value varies in different nodes. We further use *Fix Exponential* and *Fix Uniform* contributions. Given different random seeds (e.g., node ID) of strong signal and weak signal, the values of X and X' are calculated by the seeds and follow corresponding probability distribution (i.e., uniform, exponential). On TelosB node, the finest granularity of stable MSP430 MCU clock is 32768 Hz. According to default Box-MAC [26], we set T_s and δ to 12ms and 1ms, respectively. The CDFs of the five 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 five 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 the order. Moreover, compared with *Exponential* (Figure 7(b)), the Δ_{valid} of *Fix Exponential* (Figure 7(d)) increases much faster. No more than 70% of Δ_{valid} is less than 100000μs when T_a is larger than 2796μs. The trend is much similar in the comparison between *Uniform* (Figure 7(c)) and *Fix Uniform* (Figure 7(f)). This observation indicates that the strategy of fixed X is less efficient than varied X. The reason is that the difference between X and X' is also fixed with fixed X and X'. Either small or large fixed difference needs more time to reverse the order of overlapped signals.

Moreover, compared with *Uniform* (Figure 7(c)) and *Exponential* (Figure 7(b)), the Δ_{valid} of *Gaussian* (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(c). 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.

Next, we will carefully compare the $E(\Delta_{valid})$ between *Exponential* and *Uniform*. Since Δ_{valid} is non-negative, the relationship between $E(\Delta_{valid})$ and CDF function $F(\Delta_{valid})$ is as follow:

$$E(\Delta_{valid}) = \int_0^{\infty} (1 - F(\Delta_{valid}))d(\Delta_{valid}) \qquad (2)$$

In Figure 7, $E(\Delta_{valid})$ is equal to the area of the upper left region encompassed with CDF curve, left and upper border axes. Figure 8 shows the difference of $E(\Delta_{valid})$ between *Exponential* and *Uniform* under various of T_a. When T_a is less than 2796 μs, the $E(\Delta_{valid})$ of *Exponential* is about 33 0μs

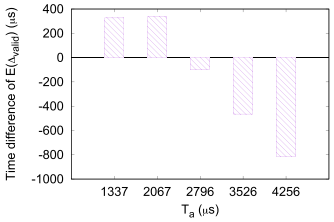


Fig. 8. The difference of $E(\Delta_{valid})$ between uniform and exponential distribution of X under different on-air time.

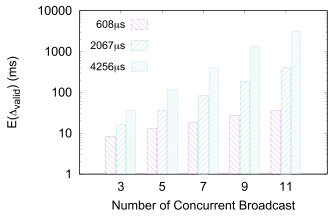


Fig. 9. The $E(\Delta_{valid})$ under different number of concurrent broadcast and different preamble packet on-air time.

smaller than *Uniform*. The $E(\Delta_{valid})$ of *Uniform* is smaller than *Exponential* when T_a is larger than 2796 μs. When T_a is 4256 μs, the $E(\Delta_{valid})$ of *Uniform* is 815 μs smaller than *Exponential*. In summary, *Chase* chooses *Exponential* distribution when T_a is smaller than 2796 μs and *Uniform* distribution for the rest of T_a.

2) Multiple-Senders Model: Two senders construct the basic concurrent transmission scenario, but there are usually more concurrent senders in real scenarios. In general, we can divide those signals into two categories, the strongest signal and other signals. We call the overlapped other signals as "weak signal". Due to the extreme diversity of "weak signal", it is hard to put the timing of "weak signal"into a nutshell. However, in the most conservative case, the temporal constraint between "weak signal" and the strongest signal is satisfied when temporal constraint between each independent signal of the "weak signal" and the strongest signal is satisfied. Thus, we define a multiple-senders model as the combination of multiple two-senders models. We use this model to show the lower bound of $E(\Delta_{valid})$ under multiple senders scenario.

We assume that "weak signal" contains Π signals. For the w^{th} signal, the initial time offset is θ_w and k'_w preamble packets are transmitted during Δ_{valid}. Moreover, we use $T'^{i}_{ippi}(w)$ to depict the i^{th} IPPI between the i^{th} and $(i+1)^{th}$ preamble packets. $X'_i(w)$ denote the corresponding random integer of $T'^{i}_{ippi}(w)$. Base on two-senders model, we have

$$
\begin{aligned}
&\textbf{min } E(\Delta_{valid}) \\
&\quad \text{where } \Delta_{valid} = kT_a + \sum_{i=1}^{k} T^i_{ippi} \\
&\textbf{s.t. } T_a \in [574, 4256]\mu s && 1 \\
&\quad \forall w \in [1,\Pi], \quad \theta_w \sim \text{uniform}((160, T_a]) && 2 \\
&\quad \forall w \in [1,\Pi], \quad (k'_w - k)T_a + \sum_{j=1}^{k'_w} T'^{j}_{ippi}(w) \\
&\quad -\theta - \sum_{i=1}^{k} T^i_{ippi} + 160 \geq 0 && 3 \\
&\quad \forall w \in [1,\Pi], \quad (k - k'_w)T_a + \sum_{i=1}^{k} T^i_{ippi} \\
&\quad - \sum_{j=1}^{k'_w - 1} T'^{j}_{ippi}(w) + \theta \geq 160 && 4 \\
&\quad \forall w \in [1,\Pi], \quad \forall i \in [1,k], \\
&\quad T^i_{ippi}(w) = X_i / f_{clock} && 5 \\
&\quad \forall w \in [1,\Pi], \quad \forall j \in [1,k'], \\
&\quad T'^{j}_{ippi}(w) = X'_j(w) / f_{clock} && 6 \\
&\quad X, X' \sim f(X) && 7 \\
&\quad \int_0^{(T_s - \delta) f_{clock}} f(X) dX = 1 && 8
\end{aligned} \tag{3}
$$

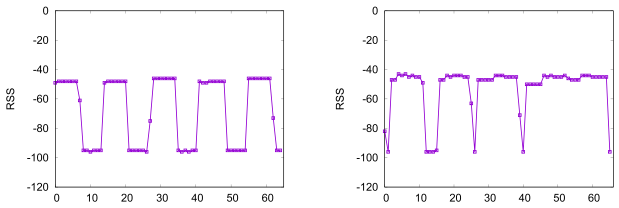
Compare with Equation 1, we renew several constraints to adapt multiple senders scenario. In constraints 3 and 4, we add temporal constraint for each signal that belongs to "weak signal". In this way, we guarantee the $(k+1)^{th}$ preamble packet of the strongest signal satisfies temporal constraint when it coexists with "weak signal".

We use the same LPL settings of Section III-B1. We set X with adaptive distribution of uniform and exponential to evaluate the $E(\Delta_{valid})$ under different number of concurrent broadcasts and different preamble packet on-air time. The results are shown in Figure 9. When T_a is small (e.g., 608 μs), a valid packet appears no later than about 36ms under 11 concurrent broadcasts. With the increasing of T_a, $E(\Delta_{valid})$ is also getting larger. When T_a is 2067 μs, $E(\Delta_{valid})$ increases to about 400ms under 11 concurrent broadcasts. With the maximum preamble packet on-air time 4256 μs, $E(\Delta_{valid})$ is about 400ms under 7 concurrent broadcast. Further, $E(\Delta_{valid})$ quickly becomes much larger than 1000ms. Thus, *Chase* should control the number of concurrent broadcasts when the length of preamble packet is close to maximum.

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 multiple preamble packets of different senders with random IPPI. More 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 should be 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



(a) (b)

Fig. 10. Illustration of the difference between RSS sequences of (a) signal transmission and (b) concurrent broadcast.

sequences of single transmission and concurrent broadcast are shown in Figure 10. 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 10(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 10(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 broadcast is not.

The second feature is variance of segment interval, which indicates 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 10(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:

$$S = \{s||r_{s-1} - Noise| < \Delta_{rss}, |r_s - Noise| \geq \Delta_{rss}\} \quad (4)$$

$$E = \{e||r_{e-1} - Noise| \geq \Delta_{rss}, |r_e - Noise| < \Delta_{rss}\} \quad (5)$$

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 $R_k = \{r_{I_S(k)}, r_{I_S(k)+1}, ..., r_{I_E(k)}\}$.

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** K equals 0. **then**
2: return FALSE.
3: **else if** $\exists i \in [1, K]$, D_i equals to 1. **then**
4: return FALSE.
5: **else if** K equals 1. **then**
6: return TRUE.
7: **else if** K equals 2 and $V_{on} < \kappa$ **then**
8: return FALSE.
9: **else if** $V_{on} < \kappa$ and $V_{segi} < \tau$. **then**
10: return FALSE.
11: **end if**
12: return TRUE.

3) Feature Extraction:

Variance of on-air time: The on-air time of the k^{th} segment is calculated as:

$$T_{on}(k) = (I_E(k) - I_S(k))\frac{1}{f_s} \quad (6)$$

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}.

$$V_{on}(R) = max\{T_{on}(i) - T_{on}(j)|1 \leq i, j \leq K\}. \quad (7)$$

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

$$SegI(k) = (I_S(k+1) - I_E(k))\frac{1}{f_s} \quad (8)$$

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.

$$V_{segi}(R) = max\{SegI(i) - SegI(j)|1 \leq i, j \leq K-1\} \quad (9)$$

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

$$D_i = \begin{cases} 0, & \text{No decoded preamble packet} \\ 1, & \text{Otherwise} \end{cases} \quad (10)$$

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 the corresponding decoded vector D, Algorithm 1 shows the process to verify whether collided broadcast packets exist. At line 1, if no segment is detected, there is no transmission.

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

At line 3, if one preamble packet is successfully received, receiver needs not extend T_t. At line 5, if only one segment is extracted, due to the window of RSSI sampling is usually longer than the length of a packet, *Chase* conservatively treats the overlapped signals as concurrent broadcast. 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 broadcast, κ and τ is close to 0. Otherwise, κ and τ is usually larger than 0, due to the diverse pattern of signal overlapping. At line 7, if there are two segments, only V_{on} is available. There is no concurrent broadcast when V_{on} is smaller than κ. At line 9, in other cases, 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 broadcast. The time and space complexity of identification process are $O(K)$.

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], [13], 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, *Chase* can quickly complete network flooding. Thus, unicast traffic can be transmitted when the channel is clear. *Chase* keeps the efficiency of those kinds of traffic as much as possible.

IV. IMPLEMENTATION

We implement *Chase* with TinyOS 2.1.2 on TelosB nodes. We use the implementation of Box-MAC LPL as the MAC protocol. Besides the default Box-MAC LPL parameters as shown in Figure 4, other parameter settings are summarized in Table I. *Chase* may benefit from better-performing LPL MAC protocols, such as ContikiMAC [7].

A. Random Function

We empirically choose the boundary T_x as 2067μs. The decision criteria of X distribution is shown in Equation 11.

$$X \sim \begin{cases} Exp(2f_{clock}/(T_s-\delta)), & 0 < T_a <= 2067 \\ Uni(0, f_{clock}(T_s-\delta)), & 2067 < T_a <= 4256 \end{cases} \quad (11)$$

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.

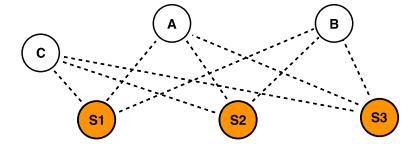
C. RSS Sampling Control

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 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.



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

Fig. 11. Illustration of the topology and link state in controlled experiments.

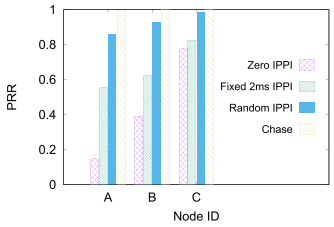


Fig. 12. PRR performance under different IPPI adjustment strategies.

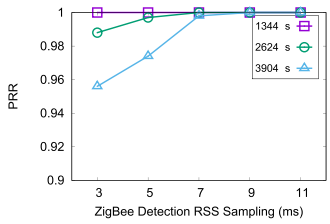


Fig. 13. PRR performance under different detection RSS sampling duration.

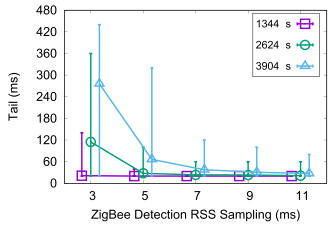


Fig. 14. Tail length under different detection RSS sampling duration.

A. Chase *Efficiency*

We use 6 TelosB nodes in control experiments. The topology and link state are shown in Figure 11. **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 11.

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. The initial phase does not appear in later testbed experiments. The initial phase is just used to guarantee concurrent broadcast in the controlled experiments. 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 [34] does.

To examine the influence of different components of *Chase* on delivery reliability, we compare *Chase* with other three concurrent broadcast strategies: (1) set IPPI as zero (default LPL), (2) set a large and fixed IPPI as 2ms (adaptive LPL for concurrent broadcast [22]), 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 12. 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 PRR of all receivers further increases and becomes larger than 85% with random IPPI. Compared with fixed IPPI, random IPPI makes different arrival time 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 11, 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 13. When T_a is 1344 μs, 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 14. 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 3904 μs, the average tail length increases to about 270ms and the maximum tail length is 440ms, about 13.6 and 22 time of 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. Number of Concurrent Broadcast

In the controlled experiments, the number of concurrent broadcast is fixed as 3. We further conduct experiments to show the impact of the number of concurrent broadcast. In the experiments, we compare *Chase* to random IPPI strategy (Section III-B) with fixed tail 100ms and 200ms. With fixed tail, the tail will not be adaptively extended by the detection of collied broadcast packets and the receiver only depends on the chance of capture effect the constructed by random IPPI strategy. T_a is set as 3584 μs. Other parameters is set as default. The senders are randomly put on a cycle, whose radius is about 20cm. The receiver is put on the centre of the cycle. Due to the heterogenous sensitivity of receiver radio in different direction, the RSS of different signals may not the same. We use the same way of Section V-A to ensure the concurrent broadcast. The power of senders is set as 7. We use the average PRR to indicate the delivery reliability of the concurrent broadcast.

The results is shown in Figure 15. We can observe that when the number of concurrent broadcast is less than 5, the PRR of all three strategies is 100%. Moreover, with 100ms fixed tail, the PRR decreases when the number of concurrent broadcast is larger than 5. In contrast, the decrease of PRR begins when the number of concurrent broadcast reaches to 8 and 10 for 200ms fixed tail and *Chase*. The PRR of 100ms fixed tail strategy drops faster than the other two. When the number of concurrent broadcast is 13, the PRR drops to 31.5%. The PRR is 84.7% with 200ms fixed tail under the same situation. With *Chase*, the PRR is not less than 97% under all situations. To conclude, we can see the random IPPI strategy is efficient when the number of concurrent broadcast is less than 5. With too many concurrent broadcast, the probability to construct capture effect is getting low and the probability of physical constrain is getting high. It needs more time to resolve the collision. Hence, the longer tail can tolerant more number of concurrent broadcast. *Chase* adaptively extents the tail so that the reliability is guaranteed. The 3% decrease on PRR of *Chase* comes from the false of identification algorithm.

C. 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 11, 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.

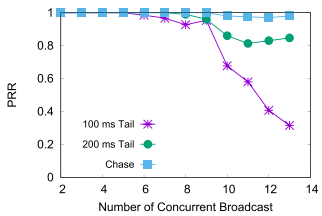


Fig. 15. The impact of the increasing of concurrent broadcast on delivery reliability.

TABLE II
THE ACCURACY OF IDENTIFICATION ALGORITHM

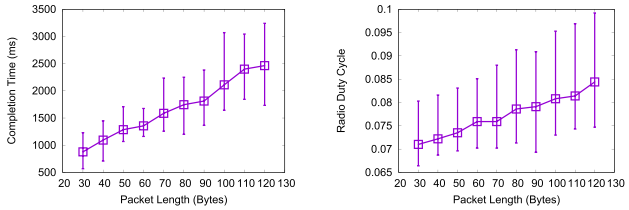
	Correct	False Negative	False Positive
Concurrent Broadcast	98.7%	1.3%	0
Single Transmission	100%	0	0
Channel Contention	99.2%	0	0.8%
Hidden Terminal	30.5%	0	69.5%

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.

D. Network Flooding

We evaluate *Chase* on two real testbeds. One testbed (called *office testbed*) have 50 TelosB nodes in our office environment. The 50 TelosB nodes are deployed as a grid topology as shown in Figure 16. The distance between two adjacent nodes of both vertical and horizontal direction is about 20cm. The other is Indriya testbed [3] with 95 TelosB nodes. We mainly use two metric to evaluate protocol performance in terms of time and

Fig. 16. The picture of real testbed with 50 TelosB nodes.



(a) (b)

Fig. 17. The completion time and radio duty cycle of Chase under different packet length on office testbed. (a) Completion Time. (b) Radio Duty Cycle.

TABLE III

THE INFLUENCE OF TAIL EXTENSION ON BOTH COMPLETION TIME AND RADIO DUTY CYCLE (RDC) ON OFFICE TESTBED

	Completion Time (ms)			Energy (RDC)		
	Min.	Avg.	Max.	Min.	Avg.	Max.
Chase	1161	1355.74	1657	7.02%	7.59%	8.51%
Chase w/o Tail Extension	1960	2849.61	4111	6.97%	7.81%	8.55%

energy. The first one is *flood completion time*, which indicates the delay from the beginning of a network flooding to all of nodes successfully receive the flooding packet. The second metric is radio duty cycle (RDC) to reflect the average energy consumption during a period.

1) Influence of Tail Extension: We evaluate the efficiency of tail extension strategy on office testbed. We set the transmission power as 2. In such transmission power, the network diameter of office testbed is about 3 hops. This is a dense network, where packet contentions and collisions may happen frequently. The flooding packet length is set as 60 bytes. Other system parameters are set as default. Node 0 (left bottom corner in Figure 16) initializes a flooding every 10 s. We calculate the average radio duty cycle every 60 s. We calculate flooding completion time for each flooding packet, then calculate the minimum, average and maximum completion time of continuous 100 flood packet.

The results are shown in Table III. We can see that the average completion time increases 110.12% without tail extension. This indicates the channel contention and collision are serious, the short tail cannot guarantee node to receive a preamble packet. It also verifies tail extension can keep node awake and quickly receive a preamble packet with random IPPI scheme. The average radio duty cycle with tail extension does not increase, but reduces 0.22%. This is because without tail extension, transmission failure leads more energy is needed to retransmit flood packet so that the radio duty cycle has a little of increment.

2) Influence of Packet Length: As our analysis of random IPPI strategy in Section III, flooding completion time will increase when the packet length is getting large. Here, we conduct experiments on office testbed to show the trend of flooding completion time increment with the increasing of packet length. We set the transmission power as 2 and use default settings of *Chase*. We increase packet length from 30 bytes to 120 bytes. For each packet length, we use the same setting of Section V-D1 to obtain the corresponding completion time and radio duty cycle.

The results are shown in Figure 17. We can see that both average completion time and radio duty cycle almost linearly increases. When packet length increases form 30 bytes to 120 bytes, the average completion time increases 180%, from 877ms to 2463ms. The average radio duty cycle increases 18.8%, from 7.1% to 8.44%. This verifies our observation of random IPPI analysis. With multiple concurrent senders, node needs more time to receive a preamble packet. This delay increase quickly with the increasing of packet length. Although the consequential tail extension further increases the radio duty cycle, it guarantee the reliability. We can further control the number of concurrent transmission to improve both completion time and radio duty cycle by delicate sender selection.

3) Dense Flooding: Here, we compare *Chase* with four other protocols on both office testbed and Indriya testbed, which are dense network environment. The first is deluge flooding with Box-MAC, in which node will immediately broadcast the received packet with duplicate suppression [16] and carrier sense. The second is the widely used flooding protocol Drip [31] with Box-MAC. In Drip, the broadcast is controlled by trickle timer [20] after receiving the packet to further reduce the influence of collision. The third is Drip with AMAC [9], which is the state-of-the-art energy efficient receiver-initiated asynchronous duty cycle MAC. The last is Contiki best effort flooding [8] with ContikiMAC [7]. Contiki flooding use polite broadcast, in which after a node receives a flooding packet, if another same flooding packet is received within a time interval, the node will not broadcast the flooding packet to reduce channel contention. We use *Chase*, LPL-Delu, LPL-Drip, AMAC-Drip and Contiki-Polite to represent these flooding protocols, respectively.

In the experiments of office testbed, we use the same settings with Section V-D1. The experiment results of office testbed are shown in Figure 18. In Figure 18(a), we can see *Chase* is the fastest flooding method. The average completion time of Chase is about 1355.74ms, which is about 29.3% faster than LPL-Delu, which is the second fastest flood method. The average completion time of LPL-Drip, AMAC-Drip and Contiki-Polite is much slower. In LPL-Delu, the constant contention backoff and packet retransmission make the completion time longer. However, the quick forward strategy and carrier sense of LPL-Delu make most of nodes quickly receive the flooding packet in dense network. For LPL-Drip and AMAC-Drip, the trickle timer increases the waiting time of flooding packet forwarding. In the worst case, only one node does not receive the flooding packet. It has to wait for a long interval. Contiki-Polite will take exponential backoff once channel

TABLE IV
THE COMPARISON OF COMPLETION TIME AND RADIO DUTY CYCLE BETWEEN CHASE AND FLOODING WITH DIFFERENT TRANSMISSION POWER ON INDRIYA TESTBED

	Tx Power 31						Tx Power 19						Tx Power 7					
	Comp Time (ms)			Energy (RDC)			Comp Time (ms)			Energy (RDC)			Comp Time (ms)			Energy (RDC)		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Chase	754	920.3	1189	8.9%	10.3%	14.2%	806	1064.5	1584	7.1%	8.6%	10.1%	1216	1649	2056	6.3%	8.9%	10.5%
LPL-Delu	1950	2799.5	4555	1.1%	5.7%	9.13%	2162	3240.8	5277	1.5%	5.9%	10.3%	3977	4974.1	6782	2.1%	7.2%	16.3%

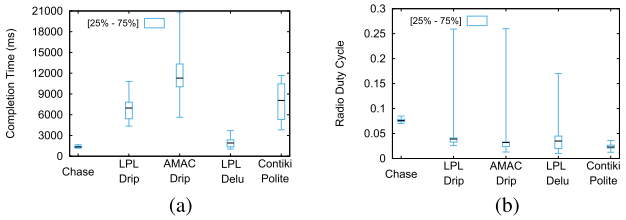


Fig. 18. The comparison of completion time and radio duty cycle among several protocols on office testbed. (a) Completion Time (b) Radio Duty Cycle.

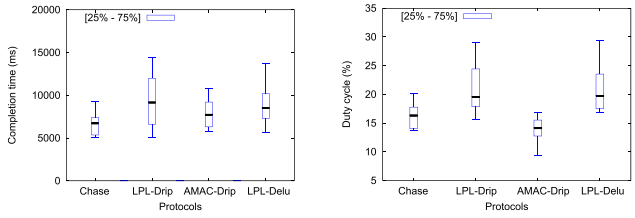


Fig. 19. The comparison of the results on office testbed experiments. (a) Distribution of the network completion time. (b) Distribution of average duty cycle.

contention is detected. Hence severe contention backoff and flooding packet retransmission of dense network flooding lead large completion time of LPL-Drip, AMAC-Drip and Contiki-Polite.

In Figure 18(b), the average duty cycle of *Chase* is 7.59%, which is about 100% higher than other flooding method. However, the maximum duty cycle of LPL-Drip, AMAC-Drip and LPL-Delu is much higher than Chase. In *Chase*, all nodes forward the received flooding packet. In such way and dense network, the tail is easily extended when node wakes up every time. Thus, the duty cycle of different nodes is close and large. For LPL-Drip and AMAC-Drip, trickle timer controls the number of flooding packet transmission of each node. For LPL-Delu and Contiki-Polite, duplicate suppression scheme can only make several nodes to forward flood packet several times and others to keep silent. In Contiki-Polite, the radio is close during channel backoff. Thus, the average energy consumption of these protocols is low.

We further compare *Chase* with LPL-Delu on Indriya testbed, which can be denser than office testbed. To achieve different network density, we change the transmission power as 31, 19 and 7, respectively. The results are shown in Table IV. We can see the average completion time of LPL-Delu is about 3 times that of *Chase*. This indicates *Chase* can capture the earliest wake-up opportunities of all nodes and make them reliably receive a preamble packet in such dense concurrent broadcast. The contention backoff and packet retransmission slow down the completion time of LPL-Delu. The average radio duty cycle of Chase is larger than LPL-Delu, but the gap becomes small with the network density gets low. With the decreasing of the number of concurrent transmission, the frequency of tail extension of *Chase* is reduced. Thus, the average radio duty cycle of *Chase* reduces from 10.3% to 8.9% with the transmission power reduces from 31 to 7. However, when network density gets low, duplicate suppression mechanism is getting inefficient. More nodes need to forward the received flood packet. Thus, the duty cycle of LPL-Delu increases 26.3% with the transmission power reduces from 31 to 7.

Overall, *Chase* is the fastest method, but not the most energy efficient due to the frequent tail extension. To achieve better energy efficiency of *Chase*, one way is to control the network density. Thus, how to reduce the number of concurrent forwarder, but keep the time efficiency is an open problem.

4) Long Hop Flooding: In contrast with dense network, we further evaluate *Chase* in long hop flooding scenario on office testbed. We set the radio output power to 3. The packet length is 77 bytes corresponding to on-air time 2623μs. Other parameters follow the settings in Section IV. To achieve more hops, we manually set the neighbors for each node. Thus, a node can only receive the packets from its assigned neighbors and drops the packets from other nodes. For example, node 49 (the right top node in Figure 16) can only receives the flood packet from the 3 around closest nodes. Thus, the network density is set as about 4 neighbors per node on average. The network diameter is about 10 hops. In this settings, a node's transmission may interference a multi-hop away node's transmission with transmission power 3. Hopefully, different hop nodes may broadcast in different time. Thus, the number of concurrent broadcast nodes gets lower.

Different with fixed period of radio duty cycle calculation, the radio duty cycle is measured during the period of each packet flooding completion time. For each protocol, Node 0 (left bottom node in Figure 16) floods 100 packets with interval 30 s to collect performance data of each flood packet. The experiment results are shown in Figure 19.

As shown in Figure 19(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 LPL-Delu 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.

As shown in Figure 19(b), the average radio duty cycle of *Chase* is smaller than LPL-Drip and LPL-Delu. The median average radio duty cycle of *Chase* is about 16.32%. The median average duty cycle of LPL-Delu 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 [9] 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.

VI. Related Work

Always-on Radio: Many approaches focus on full-coverage dissemination problem in always-on radio mode. Drip [31] and Deluge [16] are structureless with pure broadcast transmission hop by hop. They utilize the trickle timer [20] to control the dissemination flow for reducing the contention and transmission. ECD [5] further considers the influence of link quality on sender selection. CFlood [37] considers the influence of link correlation on sender selection. Cord [15] and Sprinkler [27] 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 [11] exploits the *constructive interference* to fast flooding the data in network wide. Splash [4] adopts the reverse data forwarding structure to increase reliability. Pando [6] 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, Guo *et al.* [14] 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 [28] utilizes the progress information of local neighbors to select broadcaster to increase the delivery reliability and reduce the energy consumption. Zippy [29] uses an extra always-on low-complexity transmitter and receiver hardware to quickly wake up the duty cycle radio. *Chase* enables reliable concurrent broadcast in the sender-initiated asynchronous duty cycle flooding without extra hardware.

VII. Discussion

Phase-Lock Mechanism: Phase-Lock indicates that nodes keep track of the wake-up time of their neighbors, allowing them to start the preamble just before a neighbor wakes up. With phase lock, the period of channel occupation is much shortened so that the channel contention is much alleviated. However, the potential collision cannot be reduced since it is possible multiple neighbors simultaneously send flooding packet to a node after it wakes up. Moreover, node cannot precisely know all the deliverable neighbors at the beginning. Missing any delivery opportunity may increase the network completion time. This is another kind of mishearing. In contrast, Chase can avoid any packet collision and mishearing to guarantee the fastest completion of network flooding.

Synchronous Flooding Service in Asynchronous Duty Cycle Network: pTunes [38] and CRYSTAL [17] adopts synchronous flooding service and periodically interrupt asynchronous duty cycling to do flooding. Synchronous flooding protocols [4], [6], [11] can achieve faster network completion time (tens ofms per packet) than Chase, due to no waiting time (hundreds ofms per packet) incurred by asynchronous sleep schedule. Thus, for periodical flooding, synchronous flooding protocols are definitely better than *Chase*. However, if some applications initiate flooding irregularly, periodical flooding might incur larger delay than Chase.

VIII. 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 broadcast. In case that the time and signal requirement cannot be satisfied (e.g., the signal strength difference is less than a 3db) 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.

References

[1] Z. Cao, Y. He, and Y. Liu, "L^2: Lazy forwarding in low duty cycle wireless sensor networks," in *Proc. INFOCOM*, Mar. 2012, pp. 1323–1331.

[2] D. Liu, Z. Cao, M. Hou, and Y. Zhang, "Frame counter: Achieving accurate and real-time link estimation in low power wireless sensor networks," in *Proc. IPSN*, 2016, pp. 1–20.

[3] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3D wireless sensor network testbed," in *Proc. TRIDENTCOM*, 2011, pp. 302–316.

[4] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast data dissemination with constructive interference in wireless sensor networks," in *Proc. NSDI*, 2013, pp. 269–282.

[5] W. Dong *et al.*, "Link quality aware code dissemination in wireless sensor networks," in *Proc. ICNP*, Oct. 2011, pp. 89–98.

[6] W. Du, J. C. Liando, H. Zhang, and M. Li, "When pipelines meet fountain: Fast data dissemination in wireless sensor networks," in *Proc. SenSys*, 2015, pp. 365–378.

[7] A. Dunkels, "The contikimac radio duty cycling protocol," Swedish Inst. Comput. Sci., Kista, Sweden, Tech. Rep. T2011:13, 2011.

[8] A. Dunkels, F. Österlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proc. SenSys*, Nov. 2007, pp. 335–349.

[9] 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 *Proc. SenSys*, 2010, pp. 1–14.

[10] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proc. 10th ACM Conf. Embedded Netw. Sensor Syst.*, 2012, pp. 1–14.

[11] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proc. IPSN*, Apr. 2011, pp. 73–84.

[12] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: Tracking energy in networked embedded systems," in *Proc. OSDI*, 2008, pp. 323–338.

[13] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. SenSys*, 2009, pp. 1–14.

[14] S. Guo, L. He, Y. Gu, B. Jiang, and T. He, "Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2787–2802, Nov. 2014.

[15] L. Huang and S. Setia, "CORD: Energy-efficient reliable bulk data dissemination in sensor networks," in *Proc. INFOCOM*, Apr. 2008, pp. 1247–1255.

[16] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. SenSys*, 2004, pp. 81–94.

[17] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data prediction+synchronous transmissions=ultra-low power wireless sensor networks," in *Proc. SenSys*, 2016, pp. 83–95.

[18] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proc. SenSys*, 2013, p. 1.

[19] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson, "Low power, low delay: Opportunistic routing meets duty cycling," in *Proc. IPSN*, 2012, pp. 185–196.

[20] P. A. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proc. NSDI*, 2004.

[21] D. Liu *et al.*, "Tele adjusting: Using path coding and opportunistic forwarding for remote control in WSNs," in *Proc. ICDCS*, Jun./Jul. 2015, pp. 716–725.

[22] J. Lu and K. Whitehouse, "Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks," in *Proc. INFOCOM*, Apr. 2009, pp. 2491–2499.

[23] X. Mao, X. Miao, Y. He, X.-Y. Li, and Y. Liu, "Citysee: Urban CO_2 monitoring with sensors," in *Proc. INFOCOM*, Mar. 2012, pp. 1611–1619.

[24] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. SenSys*, 2004, pp. 39–49.

[25] L. Mo *et al.*, "Canopy closure estimates with greenorbs: Sustainable sensing in the forest," in *Proc. SenSys*, 2009, pp. 99–112.

[26] D. Moss and P. Levis, "BoX-MACs: Exploiting physical and link layer boundaries in low-power networking," Comput. Syst. Lab., Stanford Univ., Stanford, CA, USA, Tech. Rep. SING-08-00, 2008.

[27] 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 Trans. Mobile Comput.*, vol. 6, no. 7, pp. 777–789, Jul. 2007.

[28] 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 *Proc. SenSys*, 2009, pp. 43–56.

[29] F. Sutton, B. Buchli, J. Beutel, and L. Thiele, "Zippy: On-demand network flooding," in *Proc. SenSys*, 2015, pp. 45–58.

[30] *TelosB*, Crossbow Inc., Milpitas, CA, USA, 2013.

[31] G. Tolle and D. E. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Proc. EWSN*, Feb. 2005, pp. 121–132.

[32] J. Wang, Z. Cao, X. Mao, and Y. Liu, "Sleep in the dins: Insomnia therapy for duty-cycled sensor networks," in *Proc. INFOCOM*, Apr./May 2014, pp. 1186–1194.

[33] T. Xiang *et al.*, "Powering indoor sensing with airflows: A trinity of energy harvesting, synchronous duty-cycling, and sensing," in *Proc. SenSys*, 2013, p. 16.

[34] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu, "Zisense: Towards interference resilient duty cycling in wireless sensor networks," in *Proc. SenSys*, 2014, pp. 119–133.

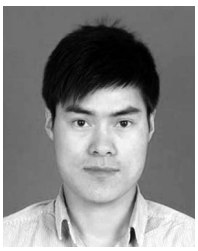
[35] X. Zheng, Z. Cao, J. Wang, Y. He, and Y. Liu, "Interference resilient duty cycling for wireless sensor networks under co-existing environments," *IEEE Trans. Commun.*, doi: 10.1109/TCOMM.2017.2692758.

[36] X. Zheng, J. Wang, W. Dong, Y. He, and Y. Liu, "Bulk data dissemination in wireless sensor networks: Analysis, implications and improvement," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1428–1439, May 2016.

[37] T. Zhu, Z. Zhong, T. He, and Z.-L. Zhang, "Exploring link correlation for efficient flooding in wireless sensor networks," in *Proc. NSDI*, 2010, pp. 1–15.

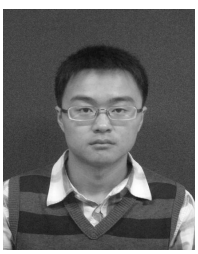
[38] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTUNES: Runtime parameter adaptation for low-power MAC protocols," in *Proc. SenSys*, Apr. 2012, pp. 173–184.

Zhichao Cao (M'12) received the B.E. degree from the Department of Computer Science and Technology, Tsinghua University, and the Ph.D. degree from the Department of Computer Science and Engineering The Hong Kong University of Science and Technology. He is currently with the TNLIST, School of Software, Tsinghua University. His research interests include sensor networks, mobile networks, and pervasive computing. He is a member of the ACM.



Daibo Liu (S'15) received the B.E. degree in computer science and technology from Dalian Maritime University, Liaoning, China, and the M.E. degree in computer science and engineering from the University of Electronic Science and Technology of China, Chengdu, China, where he is currently pursuing the Ph.D. degree. His current research interest is wireless sensor networks.

Jiliang Wang (M'10) received the B.E. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2007, and the Ph.D. degree in computer science and engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2011. He is currently with the TNLIST, School of Software, Tsinghua University, Beijing, China. His research interests include wireless sensor networks, network measurement, and pervasive computing. He is a member of the ACM.



Xiaolong Zheng received the B.E. degree from the School of Software, Dalian University of Technology, in 2011, and the Ph.D. degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, in 2015. He is currently a Post-Doctoral Researcher with the School of Software, Tsinghua University. His research interests include wireless networking and ubiquitous computing. He is a member of the ACM.