Opportunistic Concurrency: A MAC Protocol for Wireless Sensor Networks

Qiang Ma, Member, IEEE, Kebin Liu, Member, IEEE, Zhichao Cao, Member, IEEE, Tong Zhu, Member, IEEE, Xin Miao, Member, IEEE, and Yunhao Liu, Senior-Member, IEEE

Abstract—How to shorten the time for channel waiting is critical to avoid network contention. Traditional MAC protocols with CSMA often assume that a transmission must be deferred if the channel is busy, so they focus more on the optimization of serial transmission performance. Recent advances in physical layer, however, allows a receiver to reengage onto a stronger incoming signal from an ongoing transmission or interference, and thus shows the potential of parallel transmissions. Indeed, even if the channel is busy, a node has opportunities to carry out a successful transmission. In this study, we propose Opportunistic Concurrency (OPC), a new MAC layer scheme, which enables sensor nodes to capture the opportunistic concurrency and carry out parallel transmissions instead of always waiting for a clear channel. Based on local concurrency map, which encodes the interactions among different links, OPC utilizes concurrency control algorithm to make transmission decision distributedly. Our experiments on a testbed consisting of 60 TelosB sensor motes identify the transmission opportunities in WSNs with OPC. Evaluation results show that OPC achieves a 17% reduction in packet latency, a 9.4% addition in throughput and a 10% reduction in power consumption compared with existing approaches.

I. INTRODUCTION

In most of existing MAC protocols, we always hold that the data transmissions should make way for the interference. Otherwise, the transmissions must fail. The earliest collision pattern is illustrated in Fig. 1(a): If the signal of interest (SoI) arrives later than the interference, no matter how strong the power of SoI is, the signal cannot be decoded by the receivers. Carrier Sense Multiple Access (CSMA) is such a probabilistic media access control protocol common to almost all modern wireless networks. The most popular way to detect an ongoing transmission by CSMA is called "Energy Detect" [11], which is based on signal strength readings. Before transmitting, a transmitter listens to the channel. If the channel is busy, the transmission is deferred. Otherwise, the carrier sense is idle and the transmission is allowed. Obviously, to guarantee the channel quality, CSMA forbids any concurrent transmission.

In many cases the network traffic of wireless sensor networks (WSNs) becomes very heavy especially when some links suffer bursty packet forwarding tasks. To avoid traffic contention and high latency, many research efforts have been made on shortening the time for channel waiting. Most of existing approaches, however, only discuss the optimization of serial transmission performance [3], [26], [4].

The latest physical layer advances allow a receiver to reengage onto a stronger incoming signal from an ongoing interference. Capture Effect was understood through the systematic work in [13], [21]. Authors showed that the later-arrived SoI can be successfully decoded if and only if the gap is in the period of preamble (as illustrated in Fig. 1(b) and 1(c)). Message in Message (MIM) is such a physical layer capability that allows a receiver to reengage onto a stronger incoming signal from an ongoing transmission or interference even if the preamble signal has been received [14], [19].

These capabilities have been leveraged to improve throughput in enterprise wireless networks [19]. Those approaches relying on a central controller, unfortunately, are not feasible for WSNs. A sensor node must make its transmission decision based on its local information. In addition, transmission time for a packet in WSNs is usually very short. Therefore, to fulfill a concurrent transmission puts a strict constraint on the time
used for making decisions.

To address the above issues, we propose OPC, a new MAC layer algorithm integrating concurrency with CSMA to guide the transmission decisions. The main goal of OPC is to enable sensor nodes to capture the opportunistic concurrency and carry out parallel transmissions instead of always waiting for a clear channel. In Fig. 2, $S_1$ wants to forward the packets to $R_1$, while $S_2$ is transmitting packets to $R_2$. Due to the broadcasting characteristic in WSN, $R_1$ will overhear the packets from $S_2$. In this case, $S_1$ is always making way for the interference from $S_2$ under traditional CSMA protocols. In OPC, $S_1$ can selectively transmit the packets according to the SINR values on $R_1$ and $R_2$. Each sensor node maintains a local concurrency map which records the SINR values among the neighbors. Once carrier sense indicates that the current channel is busy, the node will identify the interference, then search the map to quickly find out the relevant values, and compute whether a concurrent transmission is allowed or not. Unlike traditional MAC protocols based on CSMA, OPC improves the channel by enhancing parallel transmissions.

To the best of our knowledge, OPC is the first distributed scheme which exploits concurrency to aid CSMA in making transmission decisions in WSNs. The contributions of this work are as follows.

- We present the concurrency map which encodes the interactions among different links, and thus helps sensor nodes capture the opportunities of concurrent transmissions.
- We propose a concurrency control algorithm that makes transmission decisions distributedly.
- We evaluate OPC through a real testbed consisting of 60 TelosB motes. The results show that OPC achieves a 17% reduction in packet latency, a 9.4% addition in throughput and a 10% reduction in energy consumption compared to the traditional CSMA protocol.

The rest of the paper is organized as follows. Section II gives a general background for this work, and summarizes the related work. Section III gives a series of validation experiments. Section IV presents our design and provides additional techniques to deal with several practical issues on implementation. After introducing the implementation details including interface design and overhead control in Section V, Section VI shows the evaluation results from a real indoor testbed experiments. Section VII finally concludes the paper.

II. BACKGROUND

This work is motivated by our deployed sensor network system, GreenOrbs, which aims to achieve large-scale and long-term surveillance in the forest [20], [17]. To achieve this goal, how to beneficially control and save power is critical.

A. Duty-Cycling MAC Protocol

Idle listening is one of the most significant sources of energy consumption. To enhance the energy efficiency, duty-cycling is widely used in MAC protocol for wireless sensor networks. The approaches to duty-cycling MAC protocols can be broadly divided into two categories: Techniques requiring synchronization to ensure the nodes can concurrently wake and sleep; and those that allow each node to has an independent schedule. The synchronous duty-cycling MAC protocols, such as S-MAC [28], T-MAC [26], and RMAC [4], all simplify the communication and avoid power listening, but add much complexity and need extra overhead to achieve synchronization. On the other hand, the asynchronous duty-cycling protocols consume more energy in transmitting preambles which are used to inform the neighbors to achieve action synchronization. B-MAC [24] precedes the data packet with a preamble of determined length.

In some cases, due to that the preamble can not be cut short, transmissions to the same destination have to cost much time in waiting for a clear channel. X-MAC [3] partly addresses this serial waiting problem by using a short preamble. RI-MAC [25] is similar to X-MAC, in which the sender remains active and waits silently until the receiver explicitly signifies when to start data transmission by sending a short beacon.
Delivery Ratio
Position
SoI-first SoI-last

**B. Transmission Concurrency**

Transmission concurrency has been extensively investigated in the literature. Most of them focus on 802.11. The empirical evidence of capture was showed in [13], [21], and the study in [14] analyzes the related threshold requirements. In [19] the authors make use of MIM to reorder the transmissions, thus create the concurrent transmissions. [7] allows a receiver to decode both packets from collisions, and achieves the same throughput as the collision-free scheduler. By leveraging recent works on rateless code [7], [23], the authors in [8] provides a rateless MAC design that systematically exploits interference and consistently performs well across both uplink and downlink scenarios. [2] presents a distributed algorithm based on RTS/CTS scheme, to enable simultaneous transmissions in multi-hop ad-hoc wireless networks.

In WSNs, capture awareness has been used for rapid flooding in [18] and collision detection and recovery in [27]. The authors in [22] systematically analyze the effects of combined interference. [15] presents a capture-aware linear order algorithm to estimate link interference in multi-hop wireless networks. Many existing works [6], [10], [9], [16] study about interference cancellation, which claims to compute signal of interest from interference. ZigZag [6] is an 802.11 receiver design that combats hidden terminal and contributes a new form of interference cancellation that exploits asynchrony across successive collisions. These works, however, don’t intend to change the transmission decision for the nodes. Centralized approach is not suitable in WSNs, because it is costly and unreasonable for some deployed environments. Therefore, this work designs a distributed scheme as an assistant component to help the current MAC protocol to make a decision to improve the network performance.

**III. VALIDATION EXPERIMENTS**

To better show the potential benefits, we run a series of validation experiments on ChipCon CC2420 radio [1] in an isolated environment without much interference [19]. As Fig. 4 describes, we set two transmitters *Tx* and *Intf* synchronized to each other, and broadcast their messages in the air. Two sniffers are deployed to collect ground truth. We denote SoI-first as the case when SoI comes earlier than the noise. Otherwise, SoI-last. To distinguish SoI-first and SoI-last cases and keep continuous packet transmissions, we modify the traditional MAC layer: the transmitters still turn on the carrier sense, but the result of carrier sense will not impact the transmission decision. If the channel is detected as unclear, a flag bit in the packet will be set as 1, indicating that this packet is sent out under unclear channel. By locating the receivers at different positions, we compare the number of received packets, thus analyze the relationship between SINR and packet reception ratio.

Figure 5 shows the delivery ratios for both SoI-first and SoI-last cases at different positions of the receiver. Observe that when the receiver is at position 1 or 2 (i.e., the SINR is high), the delivery ratio is always high independent of the order between SoI and interference. However, the delivery ratio of SoI-last decreases sharply at position 3 while most of SoI-first packets are received. When we put the receiver further from *Tx* at position 4, almost none of SoI-last cases at different positions of the receiver. Observe that when the receiver is at position 1 or 2 (i.e., the SINR is high), the delivery ratio is always high independent of the order between SoI and interference. However, the delivery ratio of SoI-last decreases sharply at position 3 while most of SoI-first packets are received. When we put the receiver further from *Tx* at position 4, almost none of SoI-last cases is successfully decoded, and no more than 30% of SoI-first cases overcome the interference. Besides, we search for the possible correlated parameters, such as SINR and data rate,
and summarize that SINR condition is the most critical factor (detailed in Section IV-C). That is, a MIM-like feature supports the radio to successfully decode some SoI-last packets in high SINR condition.

IV. MAIN DESIGN

In this section, we present the design of OPC. The design goal of OPC includes efficiency and fairness. First, OPC should try to capture more opportunities of concurrent transmissions thus improve the network throughput as well as reduce the packet delivery latency. Second, OPC should make sure that the transmissions will never disturb each other “on purpose”, which means, within the local perceived knowledge, every transmission decision should consider about whether other transmissions will be impacted or not.

A. Overview

In traditional approaches, a transmission decision only depends on the channel energy detected by carrier sense. In our algorithm, we try to integrate MIM-like feature and more fine-grained interference information to make transmission decisions, thus increase the packet concurrency cases. Fig. 6 illustrates the framework of OPC and shows how OPC can be embedded into the current work. We describe the establishment of the concurrency map in Section IV-B. The concurrency control algorithm for decision making is presented in Section IV-C. We propose that the hidden terminal occurs following our pursuit of transmission concurrency. Our evaluation also proves that, however, compared to large amount of potential improvement in throughput, the harm of hidden terminal brought by OPC is quite little.

B. Concurrency Map Establishment

To help a node make timely decisions at MAC layer, locally storing the current concurrency map is needed. Otherwise, the time cost for figuring out the surrounding interference fails to meet the requirement a concurrency decision needs. Utilizing the characteristic of broadcasting in wireless network, every node extracts the signal strength value from the overhearing packets and broadcasts its own record to its neighbors. Simultaneously, every node receives the beacon messages from other nodes, filtering out the useful information to build its own local concurrency map. After a period of initialization process, each node will gain its one-hop range concurrency map, which contains all the signal strength values of links between its neighbors if these two neighbors are also in each other’s communication range.

As the link quality changes as time goes by, the SINR value in the concurrency map changes even if the transmission power is fixed for every packet and the network topology is static. OPC also provides an updating scheme for concurrency map. Consider the existence of bursty link, we did not expect to find any correlation between the latest observation and those in history. In our design, the node will update its record whenever it detects its record is different with the latest observation, then broadcasts its local record in the air to inform the neighbors.

C. Concurrency Control Algorithm Design

Based on the current local concurrency map, a node can decide whether the channel is fit for transmission or not. If carrier sense detects that the channel is clear, the transmission should be allowed. Otherwise, according to the concurrency map and SINR condition, a node can analyze whether the
packet concurrency will occur or not. If the SINR requirement is satisfied, the sender launches the transmission instead of making way for the interference. Pseudocode is presented in Algorithm 1.

Algorithm 1 Transmission decision from S to R

1: Set the max concurrency number as $C_{max}$
2: Initiate the decision strategy $D = TRUE$
3: Extract the information of interference. Denote $k$ pairs of source node ID and destination node ID by $S_1$ and $R_1$, $S_2$ and $R_2$. . $S_k$ and $R_k$.
4: if $k \geq C_{max}$ then
5: \hspace{1em} Defer!
6: end if
7: if the SINR requirement on the receiver isn’t satisfied then
8: \hspace{1em} Defer!
9: end if
10: for $i = 0$ to $k$ do
11: \hspace{1em} Denote the interference to $R_i$ from $S$ by $E_{S_i}$
12: \hspace{1em} Compute the maximum allowed interference $E_i$ for $R_i$
13: if $E_i < E_{S_i}$ then
14: \hspace{1em} $D = FALSE$; break;
15: end if
16: end for
17: if $D = TRUE$ then
18: Transmit!
19: else
20: \hspace{1em} Defer!
21: end if

In Algorithm 1, we set a parameter called max concurrency number $C_{max}$, i.e., our algorithm only permits at most $C_{max}$ concurrent transmissions. Limiting the number of concurrent packets tends to decrease the following computational overhead, thus makes sure that the decision is timely and beneficial. In fact, our evaluation in Section VI clearly clarifies that the throughput does not monotonically increase with $C_{max}$. At line 12, the maximum allowed interference $E_i$ for $R_i$ means that, if and only if the transmission power of interference on $R_i$ is larger than $E_i$, collision occurs on $R_i$. Once any neighbor will be disturbed, this concurrency transmission is not allowed.

The next step is to filter useful information from the interference, i.e., the source node ID and destination node ID. How to make this real-time information available in our algorithm is nontrivial. Because the CC2420 radio is designed to be packet-level, which means we can not extract the header information until we completely receive the packet. Neglecting the difference of propagation time, this packet-level scheme provides non-available information, i.e., the decision is for a serial packet transmission, but not for transmission concurrency. We will introduce the solution in what follows.

The third step is to check whether this transmission under an unclear channel will be successful, i.e., to compute SINR value on the receiver according to the concurrency map. If the transmission power fails to achieve the requirement, we give up the transmission concurrency.

Finally, we need to consider about whether the other transmissions will be disturbed by this decision. In our implementation, no priority value is assigned to packets, thus we didn’t allow any transmission decision that will result in any other transmission failure. Actually, what we expect is to utilize the potential opportunities to create transmission concurrency, but not to enforce any transmission with high signal strength.

1) Interference Identification: In X-MAC protocol, the senders need to broadcast additional preambles to inform the receivers to awake for oncoming packets. Every transmission usually needs to broadcast many preambles, so that a node can overhear one of them to accurately infer the possible ongoing transmissions. With a local concurrency map, what a node needs are the sender ID and receiver ID, which have been contained in X-MAC (line 3 in Algorithm 1). The related pseudocode is presented in Algorithm 2.

Algorithm 2 Interference identification

1: Initiate an empty map $M$(SenderID, ReceiverID) to record the interference pair.
2: if the channel is unclear then
3: \hspace{1em} Overhear the following packet.
4: if the packet is a preamble then
5: \hspace{1em} Extract the information from the preamble.
6: \hspace{1em} if (SenderID, ReceiverID) is not in $M$ then
7: \hspace{2em} Insert (SenderID, ReceiverID) into $M$.
8: end if
9: else if the packet is a data packet then
10: \hspace{1em} Extract the information from the packet.
11: \hspace{1em} if (SenderID, ReceiverID) is in $M$ then
12: \hspace{2em} Remove (SenderID, ReceiverID) from $M$.
13: end if
14: end if
15: end if
which means the transmission from $S_2$ to $R_2$ will finish soon. Here, assume that $S_2$ only transmits exact one data packet after broadcasting preambles. Otherwise, we can easily modify the update scheme of concurrency map in the algorithm to adapt it. Besides, to avoid the number of packet concurrency is greater than $C_{max}$, we add the number of ongoing transmissions into every packets, which consumes only one byte. In addition, when $C_{max}$ is small (e.g., $C_{max}$ is 2 or 3), we can also put the detail information about interference into the packets.

2) SINR computation: After extracting the information of interference, i.e., $S_i \rightarrow R_i$, for $i = 1, 2...k$. If the channel is clear, $S$ transmits the packet as usual. Otherwise, $S$ begins to compute the SINR on $R$ based on the concurrency map. Under an assumption of additive multiple interference and non-fading channels [19], $S$ checks that if:

$$\epsilon + \sum_{i=1}^{k} I(S_i \rightarrow R_i) \leq \frac{I(S \rightarrow R)}{10^{(\tau_1/10)}}$$

$I(S \rightarrow R)$ denotes the signal strength from $S$ to $R$, and $\tau_1$ is the SoI-last SINR threshold, i.e., if SINR is greater than $\tau_1$, then $R$ can receive packets from $S$ even if $S$ transmits later. $\epsilon$ is a backoff signal strength to compensate the error due to environmental factors. Besides, it is to enable some unmeasured interference on the receiver by non-direct neighbors of transmitter.

If this condition is satisfied, the algorithm checks whether this transmission will disturb the others. Similarly, for every pair of interference $S_i \rightarrow R_i$, $S$ checks that if:

$$\epsilon + I(S \rightarrow R_i) + \sum_{i \neq t}^{k} I(S_i \rightarrow R_t) \leq \frac{I(S_t \rightarrow R_t)}{10^{(\tau_2/10)}}$$

We allow $S$ to transmit if all the conditions are satisfied. $\tau_2$ is the SoI-first SINR threshold. To keep the transmission order, we add a value field in the data and preamble packets.

V. IMPLEMENTATION

We implement OPC based on TinyOS 2.1. This section describes OPC’s implementation details. To show that it is easy to be embedded in current MAC protocol, we firstly describe the programming interfaces of OPC. Secondly, we evaluate OPC’s implementation overhead including required storage and computational overhead to claim that OPC is indeed feasible for current system configurations.

A. Programming Interface

The OPC module provides two interfaces, i.e., IMap and TDecision (in Fig. 8). IMap is for concurrency map which is the basis of the algorithm, while TDecision implements the core computation to provide transmission decisions.

1) IMap: IMap is provided to establish local concurrency map, including the signal strength values between surrounding neighbors. setNeighborSize and getNeighborSize are two commands for setting and obtaining the maximal number of neighbors. To balance the tradeoff between local storage and algorithm accuracy, the size of concurrency map should be set according to the real topology and deployment density. The sendBeacon command is used to broadcast beacon messages for the others to record the signal strength of this unidirectional link (i.e., we consider about the general cases of asymmetric link). The event beaconReceive will be signaled when the beacon message is actually received. Similarly, when the period for beacon transmission is over, through the command sendMap, we let every node broadcast its own record which contains the link signal strengths from others. Hence we signal mapReceive event to combine these map messages from the neighbors to establish a local concurrency map. Once we offer the sender ID and receiver ID, the command get is able to obtain the unidirectional signal strength value.

2) TDecision: TDecision is provided to make transmission decisions under an unclear channel condition. The command updateInterference is to update the current interference information, including different pairs of sender and receiver. If OPC is enabled, when the channel is detected to be engaged, we use command get to check whether the transmission concurrency is allowed or not. To be more specific, in the implementation of get, we repeat calling IMap.get to fetch the signal strength values between the corresponding nodes, then compute the SINR conditions. Generally, the command TDecision.get is called to make transmission decisions. If the output is TRUE, the concurrency condition is satisfied, otherwise failed.
B. Overhead

This section analyzes OPC’s implementation overhead in terms of memory overhead and computational overhead to show that our algorithm is feasible in source constrained sensor nodes.

1) Memory Overhead: OPC incurs memory overhead on RAM and ROM respectively for data and program storage. (i) To store concurrency map consumes most data memory in OPC. Let neighborsize denotes the size of neighbor table, hence every node should keep neighborsize\(^2\) records of signal strength between these neighbors. Besides, at the very beginning of map establishment, we have to store beacon information (in event IMap.beaconReceive), which needs neighborsize records. Meanwhile, collection of beacon messages can be used as index table while IMap.get expects to find the value item in the map. To specify the sender and receiver of interference, every record contains Sender ID, Receiver ID and Signal Strength, where node ID needs 2 bytes while Signal Strength costs 1 byte. Overall, \(5(\text{neighborsize}\times\text{neighborsize})\) bytes data memory is needed. In our evaluation experiments, we always set neighborsize as 16, thus the data memory overhead equals to 1.33KB, which is small compared to 10KB RAM in TelosB. (ii) To evaluate OPC’s ROM overhead, we firstly implement a simple application using CTP with the default CSMA protocol. Then, we compare this benchmark and the same one using additional interfaces in Section V-A. The original benchmark consumes 20442 bytes ROM while the modified version consumes 28012 bytes. This indicates the OPC module consumes approximately 7.4KB ROM, which is acceptable compared to 48KB ROM in TelosB.

2) Computational Overhead: OPC incurs computational overhead mainly in command TDecision.get. To be more specific, in TDecision.get, we repeatedly call IMap.get to search for the signal strength items in the map. As what described in Section V-A, the size of map equals to neighborsize\(^2\). With the index table of size neighborsize, every search operation at most checks \(2^\text{neighborsize}\) records. Overall, every search operation consumes at most 88\(\mu\)s when neighborsize equals to 16. Usually, a transmission for data packet of size 40 bytes consumes about 2ms, so the computational overhead is feasible to create transmission concurrency.

VI. Evaluation

We evaluate OPC through a real indoor testbed consisting of 60 TelosB motes (see Fig. 9) running CTP protocol, where CTP is a data collection protocol that dynamically selects the best route to the sink according to a hybrid link estimation algorithm [5]. We implement a CTP application similar to TestNetworkLpl, and the MAC protocol implements X-MAC to guide the nodes when to wake up and sleep. The evaluation will compare three MAC schemes: CSMA, OPC and NON-CSMA (i.e., always disabling carrier sense), to show that OPC is a practical design for the tradeoff between keeping out of the interference and transmitting concurrently. As we know, CSMA totally blocks transmission concurrency, while NON-CSMA omits the carrier sense procedure before a transmission.

OPC is between these two extreme protocols, opportunistically choosing to transmit under interference. We design a series of experiments to show how OPC can be utilized to improve the performance of existing sensor network protocols and applications in terms of latency, throughput and power consumption. In addition, we also validate the potential opportunities of transmission concurrency in such a real testbed system.

The transmission power of every packet is fixed as 2 and the retransmission count is 4, which means a sender at most sends a packet for 5 times. All nodes have their own sleep periods on the basis of X-MAC. Preamble length is 500ms. Note that, OPC also works with the other state-of-the-art preamble-based MAC protocols. In every experiment, we discard the data in the first 20 minutes warmup time. Due to the observation that CTP usually needs some time to achieve reliable routing, we keep each experiment running for at least 1 hour. The SINR thresholds of SoI-last and SoI-first are set as 8dB and 3dB, respectively.

A. Basic Observation

We propose that our algorithm is needed, by showing that many transmissions will be blocked by other preamble or data packets. The experiment is repeated with a varying packet transmission interval, 5s, 2s and 1s. As shown in Fig. 10, there exist 8% ~ 33% of intended transmissions blocked by an unclear channel. The percentage monotonically increases following transmission frequency and network scale. Significantly, when the data rate becomes one packet per second in a 60-node network, more than 30% of transmissions...
Fig. 10: Potential benefit of OPC.

Fig. 11: Concurrency-time. Every black dot means a transmission concurrency. X-axis represents the time series, and y-axis denotes the node ID.

will be deferred at least once, which means OPC can avoid at least 10% of deferring transmissions if it works even only in one third of these opportunities. OPC gains more improvement to those packets needed to defer for many times. Actually, we didn’t just increase the data rate in the network to enhance the opportunities of concurrent transmissions. The packet reception ratio is maintained reliably, i.e., at least 75% of motes can flow the messages to the sink.

Fig. 11 describes how often the concurrency cases happen in our algorithm. How to count concurrent transmissions in OPC? In Fig. 6, the right part explains how OPC is embedded into current framework. Before every transmission, an event called InitialBackoff or CongestionBackoff is triggered, if the result TDecision.get() shows idle, this packet is allowed to send out. There are two possibilities in TDecision.get() = idle. One is that the result of CCA shows an idle channel; the other is that the channel is busy but concurrent transmission is allowed. The former part is in all the CSMA-based protocols, and the latter is implemented in OPC. So we focus on the transmissions under the latter kind of cases, where it counts the total number of concurrent transmissions. Here shows a period of 10 minutes. In fact, we find that the distribution is even dispersing, which is reasonable because X-MAC allows each node has an independent time schedule.

Fig. 12 plots the mote locations in our testbed, and the z-axis value represents the concurrency frequency. In our observation, most of the border nodes catch more opportunities of transmission concurrency, because they are more likely to transmit beyond weak interference and avoid hidden terminals, while the inner ones always suffer more complex and strong interference. In this figure, we also find that only 8% of deployed motes never benefit from OPC (including the nodes absent in the collection).

In addition, for each packet, we record the number of times it benefits from OPC. For a 4-hop packet, it needs to be forwarded for 4 times, which means it can benefit from OPC for 4 times at most. In Fig. 13, we divide these 4-hop packets into 4 groups according to how many times they benefit from OPC. The result shows that some packets are forwarded under an unclear channel for all 4 hops, which also validates that our algorithm significantly decreases some packets’ delay. Actually, in our experiments, every 4-hop packet is forwarded under a busy channel at least once. Notably, almost every packet further than 2 hops benefits from OPC on the path to the sink.

B. Energy Consumption

In wireless sensor networks, a critical resource is energy. To reflect the performance of energy consumption, we record
For every 4-hop packet, it needs to be forwarded for 4 times, which means it can benefit from OPC for 4 times at most.

the total time when radio is on. Fig. 14 presents that to complete the same collection task, OPC always costs less power. Following the network scale alternates, the reduction of duty cycle is from 5% to 10% compared to CSMA, from 18% to 33% compared to NON-CSMA. Meanwhile, OPC slightly increases the retransmission cases. Compared to CSMA, OPC needs 5% more transmissions. It is easy to understand: 1) we encourage the nodes to create transmission concurrency only within the local interference knowledge, so hidden terminal problem is out of our consideration; 2) the interference may be not totally correctly estimated; 3) our SINR thresholds may be not accurately determined.

C. Latency

We analyze the performance of packet latency to show that our algorithm indeed decreases the end-to-end delay. As what mentioned above, our work is under an asynchronous background, thus we need some extra effort to calculate the end-to-end delay. In our experiments, the base station sends time beacons to the motes through serial ports. The motes only make use of these beacons to revise the timestamp in packets, but do not cooperate to achieve network synchronization. To comprehensively present the performance on all sides, we analyze the variations of latency in different network scales, data rates and $C_{\text{max}}$ respectively.

Fig. 15(a) presents the latency in different network scales of 20, 40 and 60 nodes. Each node transmits 60 packets per minute. As the network scale expands, more multi-hop packets may suffer more inherent delay of duty cycle. OPC improves 9% in 20-node scale, while 17% in 60-node scale compared to CSMA. Notably, it gains much more improvement compared to NON-CSMA. Fig. 15(b) presents the latency with varying data rates. As mentioned above, high data rate can create more opportunities of concurrent transmissions (though sometimes aggravates hidden terminal problem). When the data rate is low, since a data transmission only costs no more than $4\text{ms}$, thus few concurrent transmissions exist. Fig. 15(c) presents the latency under different max concurrency number $C_{\text{max}}$ of 1, 2, 3 and 4 (i.e., CSMA, OPC-2, OPC-3 and OPC-4 in Fig. 15(c) and 15(f)). When we tolerate more concurrent transmissions, the performance adversely decreases. It is mainly caused by information asymmetry between different transmitters. In our recent work [12], more concurrent transmission decisions suffer more unexpected interference from hidden terminals.

D. Throughput

To show that our algorithm indeed improves the transmission opportunities, we further investigate the performance of throughput (i.e., the average number of successful transmissions per unit time). Fig. 15(d) presents the average throughput in different network scales of 20, 40 and 60 nodes. Fig. 15(e) presents the average throughput with varying data rates. Fig. 15(f) presents the average throughput under different max concurrency number $C_{\text{max}}$ of 1, 2, 3 and 4. Similarly, when the network scale expands or data rate increases, OPC creates more concurrent transmissions, thus increases the performance compared to CSMA and NON-CSMA. Specifically, OPC achieves an 9.4% addition in throughput. In practice, however, when we allow 3 or 4 transmissions concurrently proceed, collision mostly happens.

VII. Conclusion

Traditional MAC protocols with CSMA strictly forbid concurrent transmissions. This paper presents OPC, a new MAC protocol exploiting concurrency to encourage the nodes to create concurrent transmissions. OPC is an assistant component that helps current MAC protocol make transmission decisions to improve network performance. In our decision making
scheme, each node seeks the opportunities of concurrent transmissions based on a local concurrency map, even though the channel is detected as unclear. We implement OPC based on TinyOS 2.1. Real indoor testbed experiments running CTP application presents greatly potential benefits with OPC, and further shows that OPC actually achieves a 9.4% addition in application compared to traditional CSMA protocols.

Fig. 15: End-to-end latency and throughput.

REFERENCES


Zhichao Cao received his BS degree in Department of Computer Science and Technology from Tsinghua University, China, in 2005, and his Ph.D. degree in Department of Computer Science and Engineering from the Hong Kong University of Science and Technology in 2013. He is now a postdoc researcher in the School of Software, Tsinghua University, China. His research interests include sensor networking, network routing and measurement.

Qiang Ma received his BS degree in Department of Computer Science and Technology from Tsinghua University, China, in 2009, and his Ph.D. degree in Department of Computer Science and Engineering at the Hong Kong University of Science and Technology in 2013. He is now a postdoc researcher in the School of Software, Tsinghua University, China. His research interests include sensor networks, network diagnosis and management and mobile computing.

Kebin Liu received his BS degree in Department of Computer Science from Tongji University, in 2004, and the MS and Ph.D. degrees in Shanghai Jiaotong University, in 2007 and 2010. He is currently an assistant researcher in the School of Software and TNLIST, Tsinghua University. His research interests include sensor networks and distributed systems.

Tong Zhu received his BS degree in Department of Computer Science and Technology from Nanjing University, China, and his Ph.D. degree in Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. His research interests include sensor networking, measurement, and mobile computing.

Xin Miao received his BS degree in Department of Computer Science and Technology from Tsinghua University, China, in 2005, and his Ph.D. degree in Department of Computer Science and Engineering from the Hong Kong University of Science and Technology in 2013. He is now a postdoc researcher in the School of Software, Tsinghua University, China. His research interests include sensor networks and RFID.

Yunhao Liu received his BS degree in Automation Department from Tsinghua University, China, in 1995, and an MS and a Ph.D. degree in the Department of Computer Science and Engineering at Michigan State University in 2003 and 2004, respectively. He is now Chang Jiang Professor and Dean of School of Software, Tsinghua University, China.

Zhichao Cao received his BS degree in Department of Computer Science and Technology from Tsinghua University, China, in 2005, and his Ph.D. degree in Department of Computer Science and Engineering from the Hong Kong University of Science and Technology in 2013. He is now a postdoc researcher in the School of Software, Tsinghua University, China. His research interests include sensor networking, network routing and measurement.