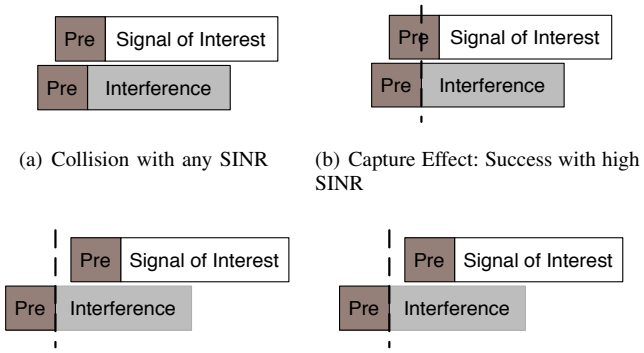
Opportunistic Concurrency: A MAC Protocol for Wireless Sensor Networks

Qiang Ma[1], Kebin Liu[1,2], Xin Miao[1], Yunhao Liu[1,2]
1 Department of Computer Science and Engineering, Hong Kong University of Science and Technology
2 TNLIST, School of Software, Tsinghua University
{maq, kebin, miao, liu}@cse.ust.hk

***Abstract*—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 like Message in Message (MIM), 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 still 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. 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.**



(a) Collision with any SINR

(b) Capture Effect: Success with high SINR

(c) Capture Effect: Collision with any SINR

(d) MIM: Success with high SINR

Fig. 1. Collision Pattern. (a): In the earliest collision pattern, no matter how strong the signal strength is, signal of interest must be lost under interference. (b)(c): Capture Effect allows SoI to be received if it is not later than a preamble's time and SINR is high enough. (d): MIM allows SoI to be received, provided SINR is high enough.

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" [5], 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, in order to guarantee the channel quality, CSMA is strict that it 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 jobs. 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 [1], [16], [2]. The latest physical layer advances allow a receiver to reengage onto a stronger incoming signal from an ongoing interference which shows the potential of parallel transmissions. These capabilities have been leveraged to improve throughput in enterprise wireless networks [10]. In practice, unfortunately, those approaches, often relying on a central controller, 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 Message in Message (MIM) 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. 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 firstly identify the interference, then check the map to quickly find out the relevant values, and compute whether a concurrent transmission is allowable or

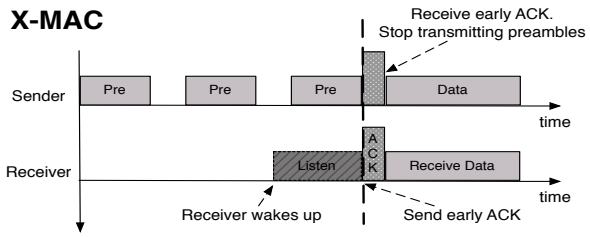


Fig. 2. X-MAC protocol. X-MAC guides the sensor nodes when to wake up and sleep. Its default transmission strategy is on the basis of CSMA, that is, no concurrency transmission is allowed.

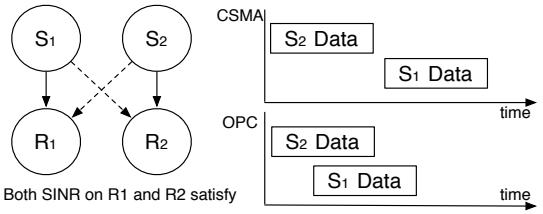


Fig. 3. Transmission opportunity. Real line like $S_1 \rightarrow R_1$ means SoI while dash line like $S_1 \rightarrow R_2$ means interference. Under CSMA, the data from S_1 must be delayed until S_2 finishes its transmission, while OPC allows transmission concurrency if both SINR on R_1 and R_2 satisfy.

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 MIM to aid CSMA in making transmission decisions in WSNs. The contributions of this works 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. Section III presents our design and provides additional techniques to deal with several practical issues on implementation. After introducing our implementation details including interface design and overhead control in Section IV, Section V shows the performance evaluation results from a real indoor testbed experiments. Section VI summarizes the related work and Section VII finally concludes the paper.

II. Background

This work is motivated by our recently deployed sensor network system, GreenOrbs, which aims to achieve large-scale and long-term surveillance in the forest [11]. To achieve this goal, how to beneficially control and save power is crucial and critical. It is well known that idle listening consumes a lot of energy [15]. Hence, low duty cycle schemes are exploited which let the sensor nodes turn on the radio when needed. One widely used low duty cycle technique is asynchronous in which extended preamble is employed to achieve action synchronization since there is no way to know whether the neighbor has woken up. In some early asynchronous approaches, before transmitting the data packet, the sender has to send the entire long preamble even though the receiver may wake up just at the beginning of preamble. Such a predetermined preamble period wastes the receiver energy, as well as leaves the sender transmitting unnecessary messages. Even worse, when multiple transmitters intend to send packets to a same node, everyone has to keep active, waiting in the queue until the channel is clear, and then begin transmitting their entire preamble while the receiver has been waken up by the first transmitter already.

In GreenOrbs, we apply X-MAC which is an advanced asynchronous low duty cycle protocol [1] to guide the nodes wake up and sleep. As illustrated in Fig. 2, X-MAC uses short preamble packets instead of a constant stream of preamble. Thus, X-MAC inserts small gaps between these preamble packets, during which the transmitter is able to listen to the medium. The gap period enables the receiver to send an early acknowledge packet to inform the transmitter, so the sender can immediately begin transmitting the data packet upon receiving the ACK packet. Compared to the traditional protocols, X-MAC shortens the preamble period. Based on CSMA, however, X-MAC does not consider the concurrency cases. When there are a number of pairs of nodes competing for the channel, they still have to proceed in the form of serial transmitting.

In order to overcome the above mentioned limitations as well as improve the network throughput and decrease the packet latency, we seek concurrent transmissions while keeping all collided packets successfully decoded by their respective receivers. In fact, some existing physical abilities support packet reception from interferences. Capture Effect was understood through the systematic work in [6], [12]. 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)). As the preamble time window is usually very short, always in the order of μs, the benefit of exploitting Capture Effect is quite little.

Message in Message (MIM) fixes Capture Effect's time limitation problem. It 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 receiver has finished receiving the preamble signal [7], [10]. Of course, a high enough SINR value is needed. This tolerance for time potentially creates lots of opportunities of concurrent transmissions. Let us consider the example in Fig. 3. In this example, S_1 wants to forward the packets to R_1, while S_2 is transmitting packets to R_2. Due to the broadcasting

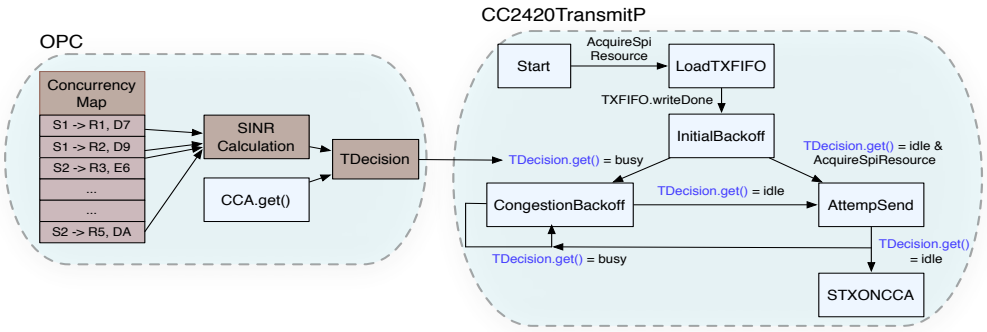


Fig. 4. Design of OPC and how OPC is embedded into current framework. In the *Concurrency Map* there are the signal strength records among the neighbors. For example, ($S_1 \rightarrow R_1$, D7) means the signal strength of the link from its neighbor S_1 to R_1 is D7, where D7 is the original raw data in TinyOS. Based on this local map, we can calculate the SINR condition on its neighbors. The final output is an interface called TDecision, which combines CCA interface in TinyOS and our calculation to decide the transmission strategy.

characteristic in WSN, R_1 will overhear the packets from S_2. In this case, if R_1 is not MIM-aware, S_1 is always making way for the interference from S_2. Otherwise S_1 can selectively transmit the packets according to the SINR values on R_1 and R_2. Why we need to consider the SINR on R_2 is that we need to guarantee the transmission from S_1 will never impact the communication between S_2 and R_2. In fact, the SINR condition on R_2 is much lower that R_1 since the transmission from S_2 is earlier than that from S_1 in our assumption. Once both the SINR values meet the requirement, R_1 can successfully decode the signal from S_1, simultaneously ensuring that R_2 can also receive the packets from S_2, thus improves the whole throughput.

III. Main Design

In this section, we present the design of OPC. The design goal of OPC includes efficiency and fairness. Firstly, OPC should try to capture more opportunities of concurrent transmissions thus improve the network throughput as well as reduce the packet delivery latency. Secondly, 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 the traditional approaches, a transmission decision only depends on the channel energy detected by carrier sense. In our algorithm, we try to integrate MIM and more fine-grained interference information to make the transmission decisions, thus increase the packet concurrency cases. Fig. 4 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 III-B. The *concurrency control* algorithm for decision making is presented in Section III-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 the node make a timely decision at MAC layer, locally storing the current *concurrency map* is needed. Otherwise the concurrency must be lost if every transmission decision is after a certain query. Utilizing the characteristic of broadcasting in wireless network, we let every node extract the signal strength value from the overhearing packets and broadcast its own record to its neighbors. Simultaneously, every node should receive 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, 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*, the 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 about MIM, we can analyze whether the packet concurrency will occur or not. If the map meets the SINR requirement and guarantees that the transmission will not disturb others, we can still launch 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 ≥ C_max then
5:     Defer!
6: end if
7: if the SINR requirement on the receiver isn't satisfied then
8:     Defer!
9: end if
10: for i = 0 to k do
11:     Denote the energy interference to R_i from S by E_{S_i}.
12:     Compute the available interference gap E_i for R_i.
13:     if E_i < E_{S_i} then
14:         D = FALSE; break;
15:     end if
16: end for
17: if D = TRUE then
18:     Transmit!
19: else
20:     Defer!
21: end if
```

In Algorithm 1, firstly we need to 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 V clearly clarifies that the throughput does not monotonically increase with C_{max}.

The next step is to gain the 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 received the packet. Neglecting the difference of propagation time, this packet-level scheme can only provide 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 and quit.

Finally we need to consider about whether the other transmissions will be disturbed by this decision. In our implementation, we didn't put any priority value in the 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.

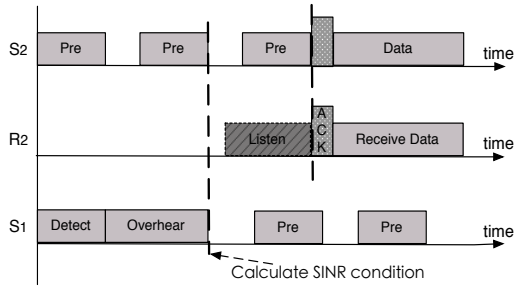


Fig. 5. Interference Identification. The main idea is to make use of preambles to timely decodes the interference information including the sender and receiver, then search the signal strength in *concurrency map* to decide whether the transmission concurrency is allowed or not.

1) Interference Identification: In X-MAC protocol, the senders need to broadcast additional preamble (small-size packet) to inform the receivers to be awake for coming transmissions. Every transmission usually needs to broadcast the preamble for many times so that we can use one of them to accurately infer the possible ongoing transmissions. because we have established a local *concurrency map*, what we need is only the sender ID and receiver ID, which have been contained in X-MAC (line 3 in Algorithm 1).

For example, in Fig. 5, when S_1 intends to send a packet to R_1, the carrier sense indicates that the channel is engaged by others. Then S_1 can overhear the ongoing transmissions in the air. If the packet is a preamble from S_2 to R_2, thus S_1 can infer that the following interference is from the link $S_2 \rightarrow R_2$, else if the packet is a data packet, which means the transmission from S_2 to R_2 is over. Here we 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 can transmit 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 [10], S can check that if:

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

$I(S \rightarrow R)$ denotes the signal strength from S to R, and τ_1 is the SoI-last SINR threshold in MIM, i.e., if SINR is greater than τ_1, then R can receive packets from S even if S transmits later. ϵ is a backoff signal strength to compensate the error.

If this condition is satisfied, the algorithm continuously check whether this transmission will disturb the others. Sim-

ilarly, for every pair of interference $S_t \rightarrow R_t$, S checks that if:

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

We allow S to transmit if all the conditions are satisfied. τ_2 is the SoI-first SINR threshold in MIM. To keep the transmission order, we add a value field in the data and preamble packets.

IV. Implementation

We implement OPC based on TinyOS 2.1. In this section, we describe OPC's implementation details. In order to show that it is easy to be embedded in the 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 our algorithm is indeed feasible for current sensor nodes (Section IV-B).

A. Programming Interface

The OPC module provides two interfaces, i.e., `IMap` and `TDecision` (in Fig. 6). `IMap` is for the *concurrency map* which is the basis of the algorithm, while `TDecision` implements the core computation to provide the critical decision.

1) IMap: `IMap` is provided to establish the local *concurrency map*, including the signal strength value between the neighbors. The two commands `setNeighborSize` and `getNeighborSize` are for the setting of maximal number of neighbors. The `sendBeacon` command is used to broadcast the 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 by the node. 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, we can use command `get` to obtain the unidirectional signal strength value.

2) TDecision: `TDecision` is provided to make the transmission decision under an unclear channel condition. The command `updateInterference` is to update the current interference information, including the different pairs of sender and receiver. If OPC is enabled, when the channel is detected to be engaged already, we use command `get` to check that 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, we only need to call the command `TDecision.get` to make the transmission decision. If the output is TRUE, the concurrency condition is satisfied, otherwise failed.

```
interface IMap {
    command void setNeighborSize(uint8_t size);
    command uint8_t getNeighborSize();
    command error_t sendBeacon(am_addr_t addr,
                               void* msg, uint16_t len);
    command error_t sendMap(am_addr_t addr,
                            void* msg, uint16_t len);
    event void beaconReceive(void* msg, uint8_t len);
    event void mapReceive(void* msg, uint8_t len);
    command uint8_t get(am_addr_t sender,
                        am_addr_t receiver);
}
```

```
interface TDecision {
    command bool get();
    command void updateInterference(
                        am_addr_t* interference_sender,
                        am_addr_t* interference_receiver);
}
```

Fig. 6. The interfaces of IMap and TDecision. Similar to TinyOS programs, the parameter *addr* means node address, *msg* is the packet, *len* is the packet length.

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 node.

1) Memory Overhead: OPC incurs memory overhead on RAM and ROM respectively for data and program storage. (i) To storage the *concurrency map* consumes the most data memory in OPC. In our design, 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 storage the 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, $5(neighborsize^2 + neighborsize)$ 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 IV-A to achieve MIM-aware CSMA. We have found that 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.

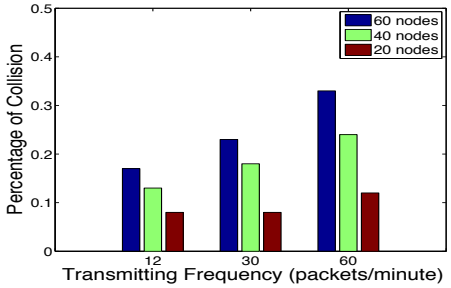


Fig. 7. Potential of OPC

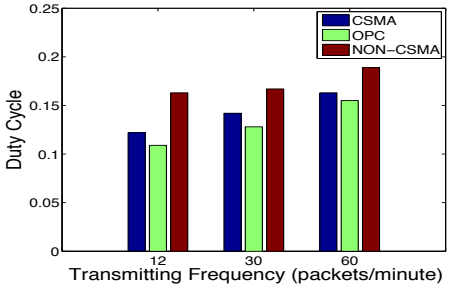


Fig. 8. Duty cycle. The time of radio on is recorded to explain the energy consumption. Here we omit the energy consumption when the radio is off since it is relatively little.

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 IV-A, the size of map equals to $neighborsize^2$. With the index table of size *neighborsize*, every search operation at most checks 2**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 for us to create transmission concurrency.

V. Evaluation

We evaluate OPC through a real indoor testbed consisting of 60 TelosB motes running the 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 [3]. We implements a CTP application similar to *TestNetworkLpl*, and the lower 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, and 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 validates the potential opportunities of transmission concurrency in such a real testbed system.

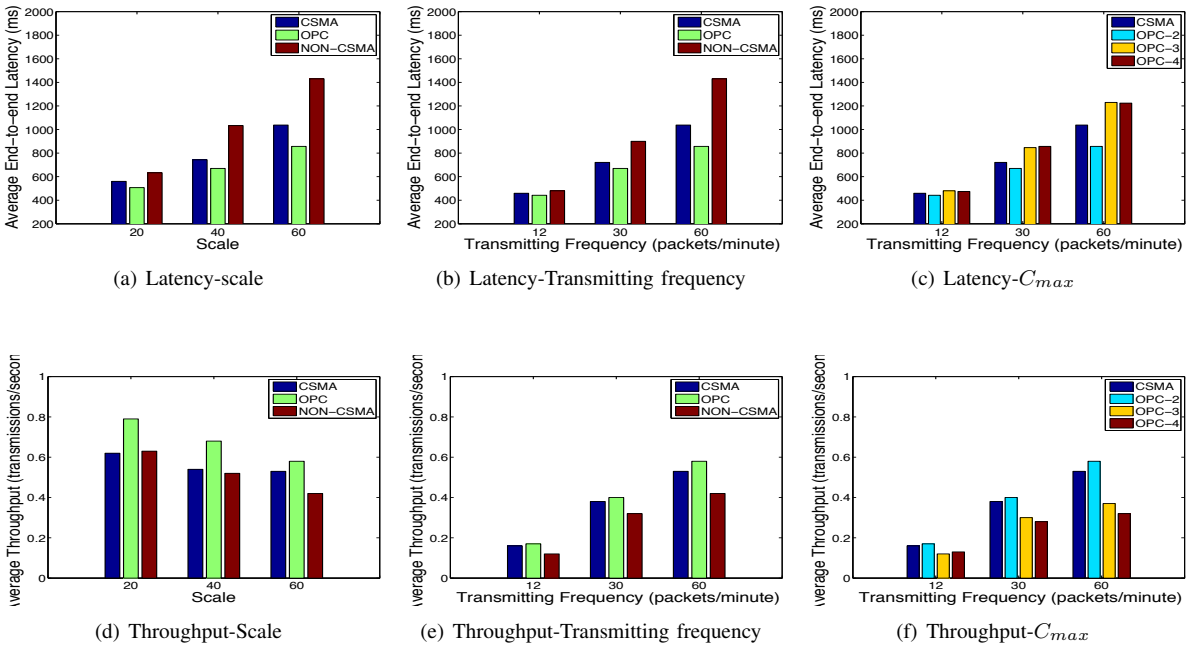
We set the transmission power is 2 and the retransmission count is 4, which means a sender at most sends a packet for 5 times. All nodes have their own sleep periods on the basis of X-MAC, and preamble length, of 500ms. In every experiment, we discard the data in the first 20 minutes warmup time. Due to the observation that CTP usually needs some time to achieve reliable routing, we keep every experiment running for at least 1 hour. To eliminate the background interference, our testbed locates in an isolated room without much interference. Besides, we set the SINR thresholds in MIM are 8dB and 3dB respectively for SoI-last and SoI-first cases.

A. Basic Observation

Firstly we propose that our algorithm is needed, due to that many transmissions will be blocked by other preamble or data packets. The experiment is repeated with a varying packet transmission interval, including 5s, 2s and 1s. In our observation (see Fig. 7), we indeed see that a large amount of transmissions have detected a unclear channel when they are to proceed, and the percentage monotonically increases with the transmission frequency and network scale. Significantly, when we set the data rate as one packet per second in a 60 nodes network, more than 30% transmissions will be deferred at least once, which means we can increase at least 14% transmissions if our algorithm works even only in one third of these opportunities. Actually we didn't just increase the data rate in the network to enhance the opportunities of concurrent transmissions. Our data is based on a certain collection ability, e.g., at least 75% of motes can flow the message to the sink through our system.

B. Energy Consumption

In wireless sensor networks, a critical resource is energy. To reflect the performance of energy consumption, we record the total time when radio is on. Fig. 8 presents that to complete the same collection task, OPC always cost 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. We also observe that OPC slightly increases the retransmission cases. Compared to CSMA, OPC needs 5% more transmissions. It is easy to understand: firstly, we encourage the nodes to create transmission concurrency only within the local interference knowledge, so hidden terminal problem is out of our consideration; secondly, the interference may be not totally correctly estimated; finally, our SINR thresholds may be not accurately determined.



(a) Latency-scale (b) Latency-Transmitting frequency (c) Latency-C_{max}

(d) Throughput-Scale (e) Throughput-Transmitting frequency (f) Throughput-C_{max}

Fig. 9. End-to-end latency and throughput

C. Latency

To delve deeply, 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, we let the base station send time beacons to the motes through serial ports. The motes only make use of these beacons to revise the timestamp when packets are sent by the nodes and arriving at the sink, 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_{max} respectively.

Fig. 9(a) presents the latency in different network scales of 20, 40 and 60 nodes, and each node will transmit 60 packets per minute. As the network scale expands, more multi-hop packets may suffer more inherent delay of duty cycle. We indeed see that OPC improves 9% in 20-node scale, while 17% in 60-node scale compared to CSMA. Notably, it is much more improvement compared to NON-CSMA. Fig. 9(b) presents the latency with varying data rates. As mentioned above, high data rate can create more opportunities of concurrent transmissions (though sometimes will aggravate hidden terminal problem). When the data rate is low, since a data transmission only costs no more than $4ms$, thus few concurrent transmissions exist. Fig. 9(c) presents the latency under different max concurrency number C_{max} of 1, 2, 3 and 4 (i.e., CSMA, OPC-2, OPC-3 and OPC-4 in Fig. 9(c) and 9(f)). As we can see, when we tolerate more concurrent transmissions, the performance adversely decreases. The reason may be manifold, like that the overlap interference may overcome the SoI, or interference identification fails and so on.

D. Throughput

In order to show that our algorithm indeed improves the transmission opportunities, thus make the nodes waste less time to defer their transmissions, we further investigate the performance of throughput (i.e., the average number of successful transmissions per unit time). Fig. 9(d) presents the average throughput in different network scales of 20, 40 and 60 nodes. Fig. 9(e) presents the average throughput with varying data rates. Fig. 9(f) presents the average throughput under different max concurrency number C_{max} of 1, 2, 3 and 4. Similarly, when the network scale expands and data rate increases, our algorithm might be used to create more concurrent transmissions, thus increase 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.

VI. Related Work

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 has an independent schedule. The synchronous duty-cycling MAC protocols, such as S-MAC [18], T-MAC [16], and RMAC [2], 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 preamble packets which are used to inform the neighbor nodes to achieve action synchronization. B-MAC [14] precedes the data packet with a preamble of determined length. In some cases, due to that the preamble can not be cut short, the transmissions to the same destination have to cost much time in waiting for a clear channel. X-MAC [1] partly addresses this serial waiting problem by using the short preamble (detailed in Section II). RI-MAC [15] 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 frame.

Characterizing Interference. In [13] the authors systematically analyze the effects of combined interference. [8] presents a capture-aware linear order algorithm to estimate link interference in multi-hop wireless networks. ZigZag [4] is an 802.11 receiver design that combats hidden terminal and contributes a new form of interference cancellation that exploits asynchrony across successive collisions.

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 [6], and the study in [7] analyzes the relates threshold requirements. In [10] the authors make use of MIM to reorder the transmissions, thus create the concurrent transmissions. In WSNs, Capture awareness has been used for rapid flooding in [9] and collision detection and recovery in [17], but they don't intend to change the transmission decision for the nodes.

VII. Conclusion

Traditional MAC protocols with CSMA strictly forbid concurrent transmissions. This paper presents OPC, a new MAC protocol exploiting MIM to encourage the nodes to create concurrent transmissions. OPC proposes that original energy detection used by CSMA should not be critical to MIM-aware receivers. In our decision making scheme, each node seeks the opportunities of concurrent transmissions based on the local *concurrency map*, even though the channel is detected as unclear. We implement OPC based on TinyOS 2.1. Real indoor testbed experiments running the CTP presents greatly potential benefits with OPC, and further shows that OPC actually achieves a 9.4% addition in network throughput, a 17% reduction in packet latency and a 10% reduction in energy consumption compared to the traditional CSMA protocol.

Acknowledgment

This work is supported in part by NSFC/RGC Joint Research Scheme N_HKUST602/08, National Basic Research Program of China (973 Program) under Grants 2010CB328000.

References

[1] M. Buettner, G.V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320. ACM, 2006.

[2] S. Du, A.K. Saha, and D.B. Johnson. RMAC: A routing-enhanced duty-cycle mac protocol for wireless sensor networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1478–1486. IEEE, 2007.

[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM, 2009.

[4] S. Gollakota and D. Katabi. Zigzag decoding: combating hidden terminals in wireless networks. *ACM SIGCOMM Computer Communication Review*, 38(4):159–170, 2008.

[5] K. Jamieson, B. Hull, A. Miu, and H. Balakrishnan. Understanding the real-world performance of carrier sense. In *ACM SIGCOMM E-WIND Workshop*. Citeseer, 2005.

[6] A. Kochut, A. Vasan, A.U. Shankar, and A. Agrawala. Sniffing out the correct physical layer capture model in 802.11 b. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, pages 252–261. IEEE, 2005.

[7] J. Lee, W. Kim, S.J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. An experimental study on the capture effect in 802.11 a networks. In *Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 19–26. ACM, 2007.

[8] J. Lee, S.J. Lee, W. Kim, D. Jo, T. Kwon, and Y. Choi. RSS-based carrier sensing and interference estimation in 802.11 wireless networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007.*

[9] J. Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *INFOCOM 2009, IEEE*, pages 2491–2499. IEEE, 2009.

[10] J. Manweiler, N. Santhapuri, S. Sen, R. Roy Choudhury, S. Nelakuditi, and K. Munagala. Order matters: transmission reordering in wireless networks. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 61–72. ACM, 2009.

[11] L. Mo, Y. He, Y. Liu, J. Zhao, S.J. Tang, X.Y. Li, and G. Dai. Canopy closure estimates with greenorbs: Sustainable sensing in the forest. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 99–112. ACM, 2009.

[12] T. Nadeem and L. Ji. Location-aware ieee 802.11 for spatial reuse enhancement. *IEEE Transactions on Mobile Computing*, pages 1171–1184, 2007.

[13] J. Padhye, S. Agarwal, V.N. Padmanabhan, L. Qiu, A. Rao, and B. Zill. Estimation of link interference in static multi-hop wireless networks. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, page 28. USENIX Association, 2005.

[14] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems, November*, pages 03–05. Citeseer, 2004.

[15] Y. Sun, O. Gurewitz, and D.B. Johnson. RI-MAC: a receiver-initiated asynchronous duty cycle MAC protocol for dynamic traffic loads in wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 1–14. ACM, 2008.

[16] T. Van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180. ACM, 2003.

[17] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. *IEEE EmNetS-II*, pages 45–52, 2005.

[18] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576. IEEE, 2002.