STAGGER: Improving Channel Utilization for Convergecast in Wireless Sensor Networks

Jiliang Wang*, Wei Dong‡, Mo Li§ and Yunhao Liu*
*MOE Key Lab for Information System Security, School of Software, TNLIST, Tsinghua University
‡ College of Computer Science, Zhejiang University
§School of Computer Engineering, Nanyang Technological University
{jiliang,yunhao}@greenorbs.com, dongw@zju.edu.cn, limo@ntu.edu.sg

***Abstract*—Channel utilization for wireless sensor networks is far from efficient, especially for convergecast in which multiple nodes are sending packets to a receiver. In this paper, we analyze the channel utilization when multiple nodes contend for the channel in convergecast and show that channel utilization can be improved by accumulating packets on each node. However, the number of accumulated packets should be carefully determined. Otherwise, the system performance may not be improved or even be degraded, e.g., incurring additional packet delay. Based on the analysis result, we present STAGGER to achieve channel utilization improvement while guarantee the worst case performance. We implement STAGGER in TinyOS 2.1 and evaluate its performance on TelosB nodes. STAGGER only uses local information to determine the number of accumulated packets without incurring additional overhead. It adopts CSMA at the low level and preserves its nice properties, e.g., fairness. The experimental results show that the design can significantly improve the per-hop throughput and reduce packet loss ratio under high traffic rate.**

I. Introduction

A wireless sensor network (WSN) typically consists of a number of sensor nodes with limited data transmission rate (e.g., 250kbps for TelosB [1] motes) contending for the wireless channel for data delivery. Thus the wireless communicational channel resource is very limited yet precious. Unfortunately, the utilization of wireless channel in WSNs is far from efficient, especially for convergecast in which multiple nodes are sending packets to a receiver. For example, the actual achievable data rate in real WSNs is even less than 50 kbps, which can hardly satisfy requirements in many applications, e.g., structural monitoring or real-time surveillance networks [2] [3]. On the other hand, convergecast is very common for WSNs. For example, multiple nodes send packets to the parent node in the data collection tree [4]. Therefore, improving channel utilization in convergecast is a stringent and important requirement for current WSNs.

The mismatch between demanding application requirement and unsatisfactory channel utilization has inspired many research works. It has been shown that unnecessary backoff is a major cause of low channel utilization for convergecast. This is especially important for WSNs since the backoff time is usually much larger than the in-air time of packet transmission. Therefore, a key idea to improve channel utilization is to reduce unnecessary backoffs while keeping a low level of collisions.

Various protocols have been proposed to address this issue. For example, [5] proposes a method to mitigate the backoff to the frequency domain. Instead of choosing randomized backoff time, nodes randomly select a subchannel for communication. However, this technique is based on OFDM radios and demand many subchannels to keep a low collision probability, which is not feasible for the current sensor hardware. Instead of resorting to the frequency domain, there are many works proposed to reduce backoff time by packet length optimization. For example, AIDA [6] and DPLC [7] offer the ability to aggregate multiple packets to a single large packet so that a single backoff time can be amortized to multiple packets. The improvement, however, is hindered by the small maximum packet size supported by current resource-limited sensor nodes (e.g., 128 bytes on CC2420 radio). There are other works, e.g., 802.11e TXOP, EDCF [8], FCR [9], which propose to send packets as a batch to amortize the per packet backoff time. While the first packet in the batch uses backoff time to mitigate collisions, backoff time of subsequent packets can be eliminated. Those works may incur additional delays due to packet accumulation and cannot guarantee the performance improvement.

To address the above mentioned issues for real WSNs, we propose STAGGER with two key features. First, it intelligently accumulates packets in order to mitigate unnecessary contention and backoff time. Second, it improves the channel utilization without introducing additional delays, which is especially important for real time WSNs. The design and implementation of such an idea is nontrivial for WSNs. We address the major challenges through three steps.

First, we revisit the limiting factors that affect channel utilization in WSNs. We theoretically analyze the channel utilization under different network parameters. The analysis provides us foundations for intelligent accumulation. Second, we make the design practically functional in a real WSN. Each sensor nodes with STAGGER distributedly make decisions based on its own traffic rate and neighborhood information. It essentially differs from existing approaches

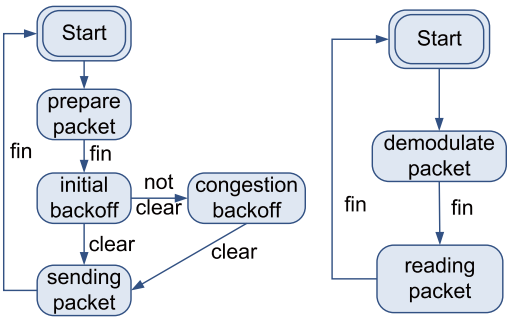


Figure 1: Sending and receiving a packet in current CSMA implementation in TinyOS.

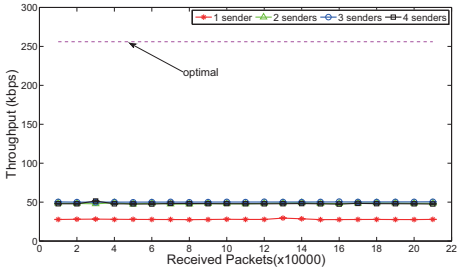


Figure 2: Throughput for different number of senders in CSMA. x-axis denotes the number of received packets at the receiver.

by guaranteeing the worst case performance. Third, we address the resource constraints of individual nodes as well as unexpected traffic patterns in WSNs. STAGGER intelligently uses the channel contending time to bootstrap packets accumulation. When there are many contentions, packets are accumulated and when there is no contention, the protocol behaves the same with CSMA.

We implement STAGGER in TinyOS 2.1 [10] [11] and evaluate it on TelosB motes with CC2420 [12] radio chip. STAGGER can be seamlessly integrated with the CSMA design at the underlying layer. By using the provided easy-to-use interfaces, users can easily port their upper layer applications to STAGGER scheme. The evaluation results show that STAGGER increases the average per-hop throughput by more than 50% while reducing the packet loss rate by more than 30% with heavy traffic loads.

The contribution of this work is two-fold. First, we theoretically analyze the channel utilization under different network parameters. Second, according to the theoretical analysis, we propose a distributed design which improves the channel utilization with delay performance guarantee. We implement the protocol and validate its effectiveness on real sensor nodes.

The rest of this paper is organized as follows. Section II presents the background of this work. Section III presents the design details. Section IV describes the implementation details of STAGGER and presents the experimental results. Section V introduces related works and Section VI concludes this work.

II. Background

We first briefly describe the main work flow of the CSMA MAC. We show that the design of CSMA is inefficient especially for the case multiple nodes are contending for the channel. This is especially exacerbated for wireless sensor networks with a small packet size. In CSMA, a node first senses the channel before transmitting the packet. If the channel is busy, the node performs backoffs until the channel becomes clear. Otherwise, the node sends the packet.

There are a variety of CSMA implementations sharing the same design principle. Figure 1 shows an example of CSMA implementation in TinyOS. To send a packet, a sender first prepares the packet, i.e., loads the packet to the on-chip buffer (TXFIFO). Then the sender performs an initial backoff. The time for initital backoff is randomly selected from the time window [0, CW_{init}). After the initial backoff, the sender begins to check the channel condition. If the channel is clear, the sender issues a command strobe (STXON) to transform the packet to radio signal and then emit the signal. Otherwise the sender repeatedly performs congestion backoffs until the channel is clear. The congestion backoff time is selected randomly from [0, CW_{con}). It can be seen that the major transmission overhead of CSMA is due to the backoff time. For example, transmitting a single packet of 35 bytes consumes only 1.1 ms while the expected initial backoff time is up to 1.5 ms in current implementation of TinyOS. The backoff time is comparable to the packet transmission time. Even without the initial backoff time, the efficiency is $\frac{1.1}{1.1+0.78} = 58.5\%$ with a single congestion backoff time of 0.78 ms. The CSMA scheme in TinyOS does not have exactly the same parameter settings with the standard. It is, however, a good representative protocol of the CSMA mechanism. It can be seen that the backoff time can significantly degrade the performance especially WSNs with a small packet size.

We further conduct a simple experiment to demonstrate that data transmission in CSMA can be very inefficient for high data rate applications. In this experiment, we let one node transmit packets of 35 bytes (including header) at its maximum sending rate to the receiver (send a packet immediately at the sendDone event in TinyOS). We measure the received data rate at the receiver. Figure 2 shows that the actual data rate at the receiver is only about 28 kbps, i.e., only 10.9% provided that the maximum PHY rate is 250 kbps for the CC2420 radio. With more senders, the data rate increases to about 50 kbps, which is still much less than the maximum data rate.

III. Protocol Design

In this section, we present the design of STAGGER to improve channel efficiency. The basic idea of STAGGER is to accumulate packets on each node and then send them in a batch. We first show the analysis of STAGGER to guarantee performance improvement. Then we show how to address practical issues in STAGGER for practical networks.

A. Protocol Analysis

In this section, we present some basic analysis of our design. Basically, STAGGER accumulates packets and sends a certain number of packets n in a batch to improve channel efficiency. By sending a batch of packets instead of sending those packets individually, a node can reduce the number of contentions for the channel. Thus the backoff time can be reduced and the channel efficiency can be improved. However, the number of accumulated packets should be carefully determined. It may also introduce additional delay for packets due to waiting time for accumulating packets. Here we analyze the impact for the number n of accumulated packet. First, we introduce two design goals of STAGGER.

G1: STAGGER should not incur additional delays to each packet. This is important as many realtime applications need packet delivery to be timeliness. More specifically, STAGGER requires that the finish time for a batch of packets is no later than that of sending those packets in the original CSMA.

G2: While G1 is satisfied, STAGGER should minimize the time used for each packet transmission in the batch so as to maximize the channel utilization.

Before present analysis results for the above two goals, we first define the following notations.

- N: We assume there are N nodes contending for a common communication channel.
- T: We assume the in-air time of a packet is T.
- λ: We assume the packet interval from the network layer to the MAC layer is λ.
- t_i: We denote the expected initial backoff time as t_i.
- t_c: We assume a single congestion backoff incurs an expected delay of t_c.

CSMA can be considered as a special case when the batch size n is equal to 1. Based on the notations, we first calculate the collision probability. We define $p(n)$ as the collision probability given the batch size of n. According to [6], $p(n)$ can be approximated as

$$p(n) = 1 - (1-\tau)^{\frac{N}{n\lambda}-1} \qquad N \geq n\lambda \tag{1}$$

where τ is the sending probability for each node. Based on the collision probability, we calculate the expected number of congestion backoffs. Denote the number of congestion backoffs as $b(n)$, we have

$$b(n) = \sum_{k=0}^{\infty} k(p(n))^k(1-p(n)) = p(n)/(1-p(n)). \tag{2}$$

Hence the total delay due to congestion backoff is $t_c \times b(n)$. We define $D(n)$ the largest delay experienced by those n packets. We have

$$D(n) = (n-1)\lambda + t_i + b(n)t_c + nT. \tag{3}$$

Accordingly, when n = 1, we have $D(1)$ as the packet delay in CSMA.

We notice that STAGGER may impact channel utilization in two different ways. First, it needs time to accumulate packets. In the worst case after accumulating n packets, a packet needs to wait for time of length $(n-1) \times \lambda$ before being transmitted. The waiting time for the first packet in the batch is increased due to the time to accumulate packets. Second, the time used for initial backoff and congestion backoff is reduced because only a single backoff is required. Such a backoff is amortized to multiple packets.

In order to satisfy G1, we require that the waiting time for the first packet in the queue should not be larger than the time without accumulation. Thus we have $D(n) \leq D(1)$, i.e,

$$(n-1) \times \lambda + t_i + t_c \times b(n) + nT \leq t_i + t_c \times b(1) + T. \tag{4}$$

This means, in the worst case, STAGGER should not introduce additional delays compared to the original CSMA.

In order to satisfy G2, we require that the time spent on each packet is minimized. In STAGGER, $D(n)$ is also the completion time of n packets. The average used time for each packet is calculated as

$$\overline{D}(n) = D(n)/n. \tag{5}$$

Therefore, our goal is to $minimize\ \overline{D}(n)$ given $D(n) \leq D(1)$.

Here we omit the details of finding the optimal n. We only show the intuitions here. There are basically two cases to choose the optimal n.

- $\lambda \leq t_i$: In this case, the packet coming rate from the upper layer is smaller than the minimal require time for each packet transmission. In such a case the sending rate is smaller than the coming rate. Accumulating more will reduce the average time per packet as long as $D(n) \leq D(1)$ holds.
- $\lambda > t_i$: This means that if time interval λ is larger than t_i, we should calculate the optimal number of accumulated packets according to equation 5. It is also possible that $n = 0$ is optimal, which means accumulating packets is not beneficial. This is reasonable when the packet coming rate is very low. In such a scenario, accumulating packets will incur a large delay.

B. Design Challenges

Based on the analysis results, we introduce the distributed design of STAGGER to improve the channel utilization. We first show the design challenges in practical WSNs. Then

we introduce the detailed design of STAGGER to conquer those challenges.

According to the analysis, each node only needs to calculate the optimal number of packets and accumulate the corresponding number of packets. However, there are several design challenges for practical networks. (1) How to distributedly approach the optimal number of packets for each node? According to the analysis in Section III-A, an intuitive method is that each sender calculates the optimal n by collecting information from other nodes. However, collecting such information incurs heavy traffic overhead, e.g., each node needs to keep updating with the number of nodes that are sending to the same receiver. Considering the scenario in presence of hidden terminals, each node even needs to acquire such information from nodes outside the neighborhood. (2) How to guarantee the performance gain for different data rates? For different data rates, the average waiting time experienced by accumulated packets becomes different. The throughput may even be reduced if the sending time is not carefully chosen. (3) How to schedule the transmissions and guarantee the fairness. Different nodes may have different number of packets to send. To guarantee the fairness, it is desirable that each node has the same opportunity to access the channel. (4) How to send the accumulated packets as quickly as possible and how to choose the sending speed since sending too fast may exceed the receiving speed and thus result in packet losses?

C. Design Overview

The design of our approach mainly consists of two components. The first component is for accumulating packets on different nodes. With the first component, different nodes distributedly make decisions on accumulating packets (Section III-D). We address challenges (1) and (2) in this component. The second component is to send multiple packets efficiently and effectively, considering constraints on wireless sensor nodes (Section III-E). We address challenges (3) and (4) in this component.

D. Accumulating Packets

In this section, we introduce the distributed protocol for accumulating packets. Each node can use its local information to determine whether or not to increase n. Meanwhile, during the accumulating process, the protocol should not increase the total waiting time comparing to the original CSMA. In order to fulfill those requirements, the component mainly consists of following steps.

1) Distributed bootstrap: At the beginning, nodes need to accumulate packets in order to reduce the congestion time. To bootstrap, during the backoff time of the first packet, the sender accumulates packets. Then the sender uses the number of accumulated packets as the initial value of n to bootstrap.

2) Increase n for different nodes distributedly: It is difficult to increase n on different nodes. Different nodes accumulate different number of packets. As the number of accumulated packets changes on some node, the optimal number of n on other nodes may also change. Here we first show a nice property of increasing n on different nodes.

Suppose there are N nodes contending the channel and each node only has the local information, we say the value n is *feasible* for node i if n satisfies inequality $D(n) \leq D(1)$ on node i. A value of n is feasible means that sending n packets will not degrade system performance. We have the following lemma.

Lemma 3.1: If n_i is feasible for node i, after increasing n on any other $N-1$ nodes, n_i is still feasible for node i.

Proof: After increasing of n on any other $N-1$ nodes, the average congestion backoff $b(n)$ experienced by node i will decrease. According to equation 3, $D(n)$ also decreases. Since $D(1)$ does not change, $D(n) \leq D(1)$ still holds and hence n_i is still feasible for node i. ■

Lemma 3.1 shows that a node can locally increase n. Increasing n on one node does not degrade the performance of other nodes. This is reasonable since accumulating packets on one node will reduce the backoff time on other nodes. Thus a node does not need to consider whether or not other nodes are increasing their values of n. This forms the foundation of our distributed protocol, in which each node can distributedly increase the value of n while not degrading other nodes' performance. This property also holds for nodes with different packet coming rate λ.

Basically, there are two conditions to stop increasing n, according to our analysis in the previous section. First, whenever reaching the optimal value of $\overline{D}(n)$, the protocol should stop increasing n. Second, the increasing process should also guarantee that the performance does not degrade comparing to when $n = 1$. Therefore, the protocol should stop increasing n when $D(n) \leq D(1)$ cannot be guaranteed. For the first condition, we simplify the protocol design by stop increasing n at the local minimum, i.e., when $\overline{D}(n) > \overline{D}(n-1)$. Thus, according to equation 3, each node needs to maintain $b(n)$. Once a node can use the channel with n accumulated packets after k backoffs, the node will update $b(n)$ as k. If no backoff counter $b(n')$ for n' is recorded, we use the value of $b(n)$ for the maximal n less than n' as a conservative approximation. For the second condition, each node checks the condition $D(n) \leq D(1)$. Each node should maintain the value for $D(1)$. Each node can record the time needed for sending the first packet and use it as $D(1)$ for the next batch.

To summarize, the protocol works as follows. First, each node accumulates packets during the backoff time of the first accumulated packets to bootstrap. Afterwards, according to lemma 3.1, each node individually increases the value of n. During the accumulation process, each node maintain $D(1)$. Each node also maintains a deadline after which

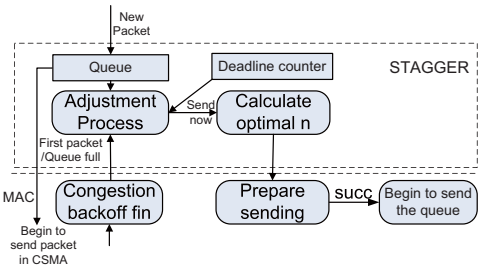


Figure 3: Integrating STAGGER with CSMA.

$D(n) > D(1)$. Upon reaching the deadline, the node will stop accumulating more packets and send the accumulated packets. Meanwhile, when $\overline{D}(n) > \overline{D}(n-1)$, the node also stops increasing n and sends the accumulated packets.

E. Sending Multiple Packets

STAGGER needs to send accumulated packet while fulfilling the following requirements. R_1: Packets are sent consecutively as quickly as possible, while not exceeding the the maximum receiving speed. R_2: Transmission of the consecutive packets is not interrupted by other nodes. R_3: fairness property should be preserved.

Before introducing the details of how to send the accumulated packets, we first study timing factors of different operations. Typically, the time for sending a packet on a sensor node under CSMA mainly consists of 3 parts: (1) $t_{prepare}$, which denotes the time for preparing the packet, e.g., loading a packet to buffer on the radio chip (e.g., TXFIFO in CC2420), (2) $t_{backoff}$, which denotes the time for initial backoff and congestion backoffs, and (3) $t_{transmit}$, which denotes the time from starting transmitting the data to finishing. The channel is only occupied by the signal for the time of $t_{transmit}$. The time $t_{prepare}$ and $t_{backoff}$ comprise the inter-packet idle time. The time for receiving a packet mainly consists of two parts: (1) $t_{demodulate}$, which is the time to demodulate the signal, and (2) $t_{process}$, which denotes the time for receiving the data, e.g., reading data from radio to the MCU.

In our design, we disable backoffs for packets in the batch except the first one. With such an approach, the sender can achieve the maximal sending rate. Since $t_{transmit}$ and $t_{demodulate}$ are almost equal, by disabling the backoff, the sending rate approaches the maximum receiving rate at the receiver, which satisfies the requirement R_1.

For the first packet in the batch, we use the CSMA mechanism for different nodes to contend for the channel. This has two merits. First, basic properties of CSMA, e.g., fairness, is preserved. When a node needs to send packets, it can contend for the channel with other nodes using CSMA. Thus requirement R_3 is satisfied. Second, batch interleaving can be avoided. Due to inter-packet idle time, a node may begin to send packets at the inter-packet idle time, resulting in packet collisions and losses. We use CSMA mechanism for the first packet in the batch. Since the inter-packet idle time is determined by packet length, each node observes the idle channel for a time period larger than the inter-packet idle time before starting a transmission. While the channel is not clear, some other node is sending and the node postpones its transmission. Thus requirement R_2 is satisfied.

Considering the impact of noise and link losses, the STAGGER design supports link layer retransmission with acknowledgement and uses a bitmap to record packet receptions. STAGGER sends acknowledgment as in CSMA without any modification. To support reliable data transmission, STAGGER also provides a `LinkRetrans` interface to upper layer for link layer retransmission. `LinkRetrans` uses the a bitmap to record unsuccessful packet transmissions and then retransmits those packets.

Example: Assume an example with two senders S1 and S2. S1 and S2 begin to contend for the channel once they have packets to send in the queue. During the initial backoff time, both S1 and S2 can accumulate packets to bootstrap. After the initial backoff and congestion backoffs, assume node S1 (S2) senses the idle channel, if S1 (S2) cannot accumulate more packets according to Section III-D, node S1(S2) sends the accumulated packets as in Section III-E and hence the other node S2(S1) will perform backoffs. Otherwise, node S1 and S2 continue to accumulate packets. As long as S1 or S2 has accumulated packets, the channel contention time will be reduced, resulting in more opportunities for both nodes to accumulate packets.

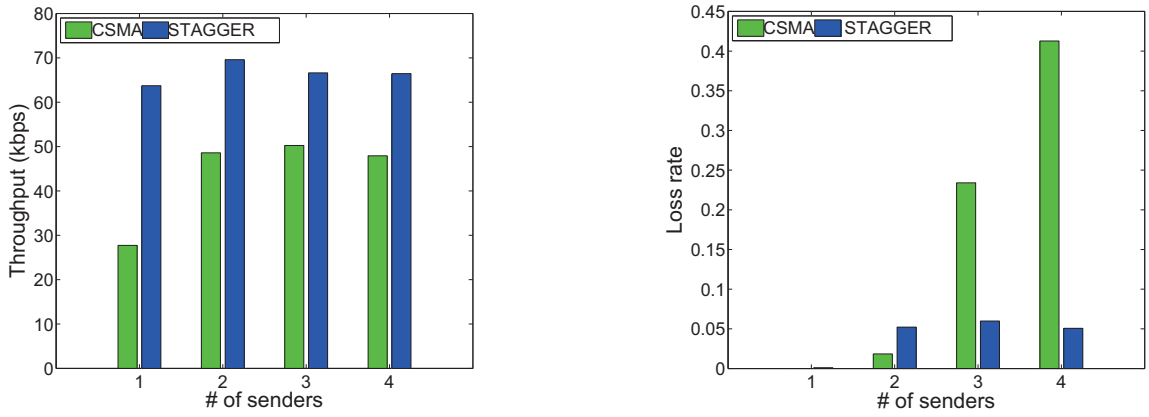
IV. Implementation and Evaluation

In this section, we present the implementation and evaluation details of STAGGER. We implement the protocol based on TinyOS 2.1 [10] [11]. The hardware platform is TelosB [1] motes with MSP430F1611 MCU and CC2420 [12] radio chip. First, we show the implementation details, including the interaction of STAGGER with other layers, programming interfaces, and overhead.

A. Implementation

1) Integrating with TinyOS MAC: The implementation of STAGGER resides between the network layer and the MAC layer. It works transparently to the upper layer. By using the interface provided by STAGGER, upper layer can send packets to STAGGER without additional operations. To interact with MAC layer, the implementation of STAGGER uses interfaces exposed by MAC layer to control the MAC layer behavior, e.g., control the backoff and fetch channel assessment value. The implementation does not need to modify the MAC layer in current CSMA.

The detailed diagram of integrating CSMA with STAGGER is shown in Figure 3. When a packet comes to the queue, STAGGER calculates the deadline according to the queue length and measures congestion backoff information



(a) (b)

Figure 4: Performance of STAGGER comparing with CSMA for different number of senders. (a) Throughput comparison. (b) Reliability comparison.

provided by MAC layer. If it is profitable to wait, STAGGER will wait for another packet according to adjustment process. Otherwise, it will disable backoff and begin to send packets in the queue. The deadline counter is used to check whether it is profitable to wait another packet to avoid a long waiting time.

2) Programming interface: One of our design goals is to provide easy to use interfaces for application. We design two components for the implementation, i.e., `STAGGERSenderC` and `STAGGERReceiverC` respectively. The `STAGGERSenderC` provides the interface `STAGGERSend`. Similar to `AMSenderC`, `STAGGERSenderC` also provides `AMPacket`, `Packet` and `PacketAcknowledgements` interfaces. The `STAGGERReceiverC` provides similar interfaces as that provided by `AMReceiverC`, which are `STAGGERReceive`, `Packet` and `AMPacket` interfaces.

3) Overhead: We evaluate the overhead of STAGGER method in terms of memory overhead and communication overhead.

- *Memory overhead*. The STAGGER method needs a queue to buffer the packets. For a queue of maximal size 10, the memory used is 10×100=1000 bytes. Considering the 10 KB RAM space for MSP430, it is acceptable to use the memory space.
- *Communication overhead*. STAGGER does not requires message exchanging among different nodes and thus does not introduce extra communication overhead.

4) Fairness: STAGGER uses the CSMA MAC provided by TinyOS. Therefore, as in CSMA, STAGGER can guarantee each node has the same opportunity to access the channel. Similar to CSMA, each node has an equal opportunity to access the channel does not necessarily mean that they use the channel equally. Nodes may use the channel for different time lengths. For example, in CSMA, a node may send a larger packet than others upon obtaining the channel. Currently, we use the same competition mechanism as in CSMA. The evaluation result shows that the probability for different nodes to use the channel is similar.

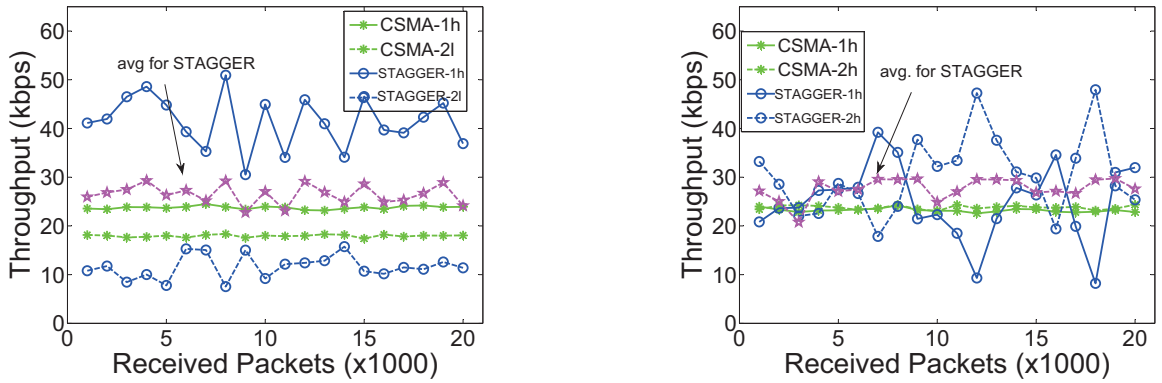
B. Evaluation

In this subsection, we evaluate the performance of STAGGER in terms of throughput and reliability. Then we test the effectiveness of STAGGER for different traffic rates and traffic patterns, e.g., random traffic, traffic with burstiness and saturated traffic. Further, we evaluate the fairness of STAGGER.

1) Throughput and reliability: We first evaluate STAGGER approach in terms of throughput and reliability. In this experiment, we use different number of senders to send packets to a receiver. We vary the number of senders to measure the throughput and packet loss ratio.

Figure 4 (a) shows achieved throughput and packet loss ratio with respect to the number of senders ranging from 1 to 4. We can see that for 1-sender case, STAGGER achieves a much higher throughput than original CSMA by reducing the unnecessary backoff time. For the case with multiple senders, senders in STAGGER still achieve a throughput gain by about 40%-60%. The throughput gain for multiple senders is because that STAGGER sends packet batches more efficiently and reduces the channel contention.

Figure 4 (b) shows the packet loss ratio for different approaches. In Figure 4 (b), we can see that for 1 or 2 senders, both STAGGER and CSMA achieve high reliability with a low packet loss ratio. As the number of senders increases, packet loss ratio for CSMA becomes higher while STAGGER still has a very low packet loss ratio. STAGGER



(a) (b)

Figure 5: Performance for senders with the different traffic pressures. 1h/2h/1l/2l denotes node ID and traffic pressure (h means high traffic pressure and l means low traffic pressure). (a) One node with high traffic and one node with low traffic pressure. (b) Both with high traffic pressure.

reduces the packet loss ratio by up to 90% for multiple senders. The improvement of reliability is because that STAGGER accumulates packets and reduces the probability of collision.

2) Impact of data rate: In this experiment, we examine the impact of data rate to the performance of STAGGER. We use two senders to send packets to a receiver with different data rates and then measure the achieved throughput for those two senders. Figure 5 (a) shows the result for two senders. One of the senders is with a high data rate and the other is with a low data rate. It can be seen that the node with a high data rate achieves a high throughput with STAGGER. The total achieved throughput at the receiver is much higher than that in CSMA.

In average, the STAGGER improves the throughput by about 15% compared to CSMA. The improvement comes from that STAGGER accumulates packets to send them together. Sending more packets together reduces the time for channel contention as well as the probability of collision. For the sender with a high data rate, the accumulated packets is more than that of the node with a low data rate, thus the throughput for the node with a high data rate is higher.

We also test the case that both two senders are with a high data rate. High data rate means a high probability of increasing the packet batch size, resulting in a throughput proportional to the data rate. As shown in Figure 5 (b), the throughput of those two senders fluctuates. This is because those two senders compete the channel using CSMA and once a node obtains the channel, it will send a batch of packets, resulting in a high throughput at that time period. For high data rate, the STAGGER improves the throughput by about 30%.

3) Impact of traffic pattern: In this subsection, we evaluate the impact of traffic pattern to the performance of STAGGER. To generate different traffic patterns, we adjust the inter-packet interval for generating packets. We generate three different traffic patterns in our evaluation: (1) normal traffic pattern, where the inter-packet interval is randomly selected from 4ms-8ms, (2) normal traffic with burstiness, where node generate bursty traffic with a probability p (p=1/8), and (3) saturated traffic, where nodes send packets as fast as they can.

Figure 6 shows the result for different traffic patterns. It can be seen that for the first two traffic patterns, STAGGER achieves a throughput gain of 50%-90%. For random traffic, the throughput gain is about 50%. The improvement of throughput mainly comes from sending multiple packets together. The throughput of saturated traffic by using STAGGER is almost two times as high as the original CSMA in TinyOS since there is a high probability that packets are batched. The throughput for bursty traffic by using STAGGER is about 90% higher than CSMA. This experiment reveals that STAGGER can be used in different traffic patterns and it is more efficient if the traffic pressure is high.

4) Fairness: In this experiment, we test the fairness property of the STAGGER. We test the fairness in two cases with 4 senders and 3 senders respectively.

Results in Figure 7 shows the Jain index values for different cases. As seen from Figure 7, each node obtains a fair channel share since the Jain Index [13] is near to 1 (A larger Jain Index indicates a higher fairness). The fairness achieved by STAGGER is because that the senders have the same probability to send a batch of packets. Therefore, the senders preserves the fairness as in CSMA.

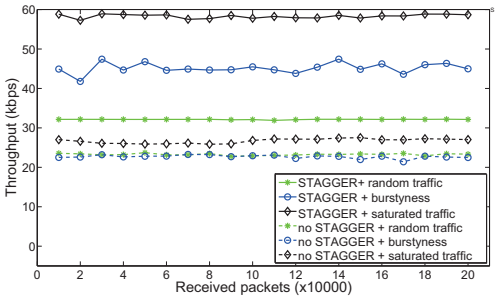


Figure 6: Performance comparison of STAGGER with CSMA under random traffic, bursty traffic and saturated traffic, respectively.

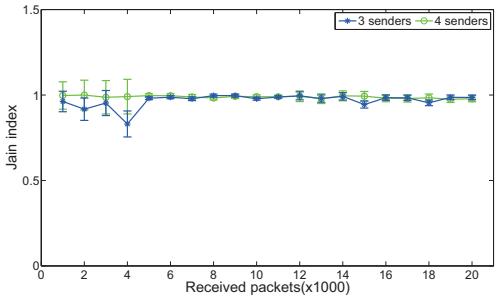


Figure 7: Fairness index for STAGGER. We test STAGGER for different senders. The Jain index values for different senders with STAGGER are nearly 1.

V. Related Work

The inefficiency of DCF, e.g., CSMA, has been noticed in wireless domain for a long time and attracted many research interests. There are a large collection of works to address this issue.

The first category is to amortize the time overhead to more sub channels. In [14], Tan. et al propose a fine grain channel access method in WLAN. Their method divides the channel to more sub-channels, each with a lower data rate. Therefore, the channel can be used in a higher granularity and the time overhead can be alleviated in each channel. However, this kind of work requires manipulation on subchannels and hence is not applicable to WSNs.

The second category is to amortize the time overhead to more nodes. Works in this category include GAMA-PS [15], CHAIN [16] and etc. Those approaches can coordinate different senders and introduce a binding relationship among different senders. In those approaches, a single backoff can incur transmissions from different senders and thus the channel efficiency is improved. Those works require cooperation between different nodes.

The third category is to amortize the time overhead to more packets. Methods in this category can further be divided into two groups. The first group aims at aggregating more packets into one to improve the channel efficiency. For example, AIDA [6] presents a method to aggregate multiple packets into a single one to reduce the average channel contention for each packet and thus can improve the throughput. Vuran et al. [17] propose a cross-layer packet length optimization method in which they consider the multihop routing and broadcast nature. DPLC [7] dynamically adapts the packet size to channel condition and finds an optimal packet size for current bit error rate. However, large packets take extra risk of collisions and suffer a higher recovery cost after collisions. Besides, the improvement for those methods is not optimized and often constrained by currently used hardware for sensor networks (128 bytes). Even for large buffer size (e.g., 1500 bytes in WLAN), the efficiency can still be improved due to high hardware speed [14]. The worst case performance of those protocols, e.g., the additional introduced delay, is not considered in those works. The second group is to send more packets in a batch to improve the channel efficiency. There are many research works in this group, such as 802.11e TXOP, EDCF [8], FCR [9], OAR [18] and CM [19]. Those approaches amortize the backoff overhead to more packets thus can improve the channel efficiency. Our work is different from those since existing approaches may increase the delay for accumulated packets [16] [20] and cannot guarantee worst case performance. Our approach calculates the optimal number of batch size and also the delay experienced by each packet to guarantee the performance improvement.

There are also many approaches aiming at improving end-to-end throughput, such as PIP [21], [22] and etc. By incorporating multi-channel technique and so on, those approaches can achieve a significant throughput improvement. Those approaches work well for particular traffic patterns, e.g., bulk data transfer with a single flow, by mitigating the setting up overhead. We design STAGGER to work as a extension to the widely used CSMA protocol and can be orthogonal to those existing protocols. Batch-and-send technique has also been used in various applications. For example, the technique has been used in different applications, e.g., volcano monitoring [23] and structure monitoring [24] [25]. STAGGER works between the MAC and NET layer and finds the optimal number of batch size to achieve improvement.

VI. Conclusion

Wireless communication for WSNs is far from efficient, especially for convergecast in which multiple nodes are contending for the channel. In this work, we first theoretically analyze the channel utilization for different network parameters, providing guidelines to improve channel efficiency. Based on the analysis results, we present a distributed protocol named STAGGER. STAGGER is essentially different from existing protocols by improving channel efficiency

while guaranteeing the delay performance. Each node with STAGGER can distributedly make decision on channel using time, while leading to channel utilization improvement. The distributed feature enables STAGGER be easily applied in WSNs. STAGGER is implemented based on TinyOS 2.1 and evaluated on TelosB motes. The evaluation results show that the design is effective in real WSNs.

ACKNOWLEDGMENTS

This work is supported in part by National Basic Research Program of China (973 Program) under grant 2011CB302705, NSFC under grant 61202359, 61202402, China Post doctoral Science Foundation under grant 2012M520013, the Fundamental Research Funds for the Central Universities (2012QNA5007), the Research Fund for the Doctoral Program of Higher Education of China (20120101120179).

REFERENCES

[1] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proceedings of ACM/IEEE IPSN*, 2005.

[2] S. N. Pakzad, G. L. Fenves, S. Kim, and D. E. Culler, "Design and implementation of scalable wireless sensor network for structural monitoring," in *J. Infrastruct. Syst*, 2008.

[3] T. B. O. Chipara, C. Lu and G.-C. Roman, "Reliable clinical monitoring using wireless sensor networks: Experience in a step-down hospital unit," in *Proceedings of ACM SenSys*, 2010.

[4] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proceedings of ACM SenSys*, 2009.

[5] S. Sen, R. R. Choudhury, and S. Nelakuditi, "No time to countdown: Migrating backoff to the frequency domain," in *Proceedings of ACM MobiCom*, 2011.

[6] He, Tian and Blum, Brian M. and Stankovic, John A. and Abdelzaher, Tarek, "Aida: Adaptive application-independent data aggregation in wireless sensor networks," *ACM Transactions Embedded Computing System (TECS)*, vol. 3, no. 2, pp. 426–457, 2004.

[7] W. Dong, C. Chen, G. Chen, Y. Liu, and J. Bu, "DPLC: Dynamic Packet Length Control in Wireless Sensor Networks," in *Proceedings of IEEE INFOCOM*, 2009.

[8] L. Romdhani and C. Bonnet, "Performance analysis and optimization of the 802.11e edca transmission opportunity (txop) mechanism," *WiMOB*, vol. 0, p. 68, 2007.

[9] Y. Kwon and Y. Fang and H. Latchman, "Design of mac protocols with fast collision resolution for wireless local area networks," *IEEE Transactions on Wireless Communications*, vol. 3, pp. 793–807, 2004.

[10] TinyOS. [Online]. Available: http://www.tinyos.net

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *Proceedings of ACM ASPLOS*, 2000.

[12] CC2420 datasheet. [Online]. Available: http://focus.ti.com/docs/prod/folders/print/cc2420.html

[13] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," in *DEC Research Report TR-301*, 1984.

[14] K. Tan, J. Fang, Y. Zhang, S. Chen, L. Shi, J. Zhang, and Y. Zhang, "Fine-grained channel access in wireless lan," in *Proceedings of ACM SIGCOMM*, 2010.

[15] E.-S. Jung and N. H. Vaidya, "An energy efficient mac protocol for wireless lans," in *Proceedings of IEEE INFOCOM*, 2002.

[16] Z. Zeng, Y. Gao, K. Tan, and P. R. Kumar, "Chain: Introducing minimum controlled coordination into random access mac," in *Proceedings of IEEE INFOCOM*, 2011.

[17] Mehmet C. Vuran, Ian F. Akyildiz, "Cross-layer packet size optimization for wireless terrestrial, underwater, and underground sensor networks," in *Proceedings of IEEE INFOCOM*, 2008.

[18] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. W. Knightly, "Oar: An opportunistic auto-rate media access protocol for ad hoc networks," *Wireless Networks*, vol. 11, pp. 39–53, 2005.

[19] Y. Xiao, "Ieee 802.11 performance enhancement via concatenation and piggyback mechanisms," *IEEE Transactions on Wireless Communications*, vol. 4, 2005.

[20] G. Sharma and A. Ganesh and P. Key, "Performance analysis of contention based medium access control protocols," *IEEE Transactions Information Theory*, vol. 55, 2009.

[21] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale, "Pip: A connection-oriented, multi-hop, multi-channel tdma-based mac for high throughput bulk transfer," in *Proceedings of ACM SenSys*, 2010.

[22] S. Duquennoy, F. Osterlind, and A. Dunkels, "Lossy links, low power, high throughput," in *Proceedings of ACM SenSys*, 2011.

[23] Geoff Werner-allen and Konrad Lorincz and Jeff Johnson and Jonathan Lees and Matt Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proceedings of USENIX OSDI*, 2006.

[24] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of ACM/IEEE IPSN*, 2007.

[25] Xu, Ning and Rangwala, Sumit and Chintalapudi, Krishna Kant and Ganesan, Deepak and Broad, Alan and Govindan, Ramesh and Estrin, Deborah, "A wireless sensor network for structural monitoring," in *Proceedings of ACM SenSys*, 2004.