CMAC: An Energy Efficient MAC Layer Protocol Using Convergent Packet Forwarding for Wireless Sensor Networks

Sha Liu, Kai-Wei Fan and Prasun Sinha
Department of Computer Science and Engineering, The Ohio State University
Email: {liusha,fank,prasun}@cse.ohio-state.edu

***Abstract*— Low duty cycle operation is critical to conserve energy in wireless sensor networks. Traditional wake-up scheduling approaches either require periodic synchronization messages or incur high packet delivery latency due to the lack of any synchronization. In this paper, we present the design of a new low duty-cycle MAC layer protocol called Convergent MAC (CMAC). CMAC avoids synchronization overhead while supporting low latency. By using zero communication when there is no traffic, CMAC allows operation at very low duty cycles. When carrying traffic, CMAC first uses anycast to wake up forwarding nodes, and then converges from route-suboptimal anycast with unsynchronized duty cycling to route-optimal unicast with synchronized scheduling. To validate our design and provide a usable module for the community, we implement CMAC in TinyOS and evaluate it on the Kansei testbed consisting of 105 XSM nodes. The results show that CMAC at 1% duty cycle significantly outperforms BMAC at 1% in terms of latency, throughput and energy efficiency. We also compare CMAC with other protocols using simulations. The results show for 1% duty cycle, CMAC exhibits similar throughput and latency as CSMA/CA using much less energy, and outperforms SMAC and GeRaF in all aspects.**

I. Introduction

Extending the lifetime of battery powered wireless sensor networks is critical because touching the sensor nodes for replacement might be expensive or even impossible. Since idle listening consumes almost the same energy as receiving or transmitting, duty cycling the radio is important to achieve long lifetime. However, low duty cycle usually incurs performance degradation in throughput and latency while they are also critical performance metrics especially in event tracking and surveillance applications. These conflicting objectives motivate the design of a new MAC layer protocol called **Convergent MAC (CMAC)**. Compared to other MAC layer protocols like BMAC [1] and SMAC [2], CMAC can significantly reduce latency and improve throughput while supporting very low duty cycles.

Current duty cycling MAC layer protocols for wireless sensor networks are either synchronized using explicit schedule exchanges or totally unsynchronized. However, both have their weaknesses and deficiencies. SMAC [2], TMAC [3] and DMAC [4] use periodic synchronization messages to schedule the duty cycling and packet transmissions. Such message exchanges consume significant energy even when no traffic is present. BMAC [1] uses unsynchronized duty cycling and uses long preamble to wake up receivers. However, the long preamble mechanism has following problems. First, the latency accumulated along multihop routes could be overwhelming due to the application of long preamble on each hop. Considering the importance of event reporting latency in tracking and surveillance applications, such latency increase is not suitable. Second, the energy consumed on preamble transmission and reception after the receiver has woken up is wasted. This is due to the lack of information at the sender side about the wake-up schedule of the receiver, and thus the preamble length is chosen conservatively. Third, other neighbor nodes of the transmitter will also be kept awake by the long preamble until the data packet transmission finishes, which is also wasteful since they are doing unneeded preamble overhearing. Polastre et. al. propose an link abstraction called Sensornet Protocol (SP) [5] to adjust the preamble length by observing recent and nearby traffic. However, SP still involves many uses of long preamble, and it cannot dynamically select next hop if the intended next hop is currently not available because of sleeping or interference.

MAC layer anycast is another way to avoid explicit synchronization [6]–[9]. However, when compared to the Contention-Based Forwarding (CBF) studied in MANET literature [10]–[17], their RTS/CTS exchange schemes are more complicated and inefficient. Both MAC layer anycast and CBF work by prioritizing the CTS replying from potential forwarders, but CBF schemes use CSMA based contention among CTS repliers to resolve CTS collisions and thus have lower overhead. The basic idea of both of them provides better packet forwarding opportunities than unicast, but none of them specify how the sleep scheduling and radio activity assessment are accomplished, which is critical for a real MAC layer protocol implementation. In addition, all of them incur higher RTS/CTS exchange overhead than unicast which could be avoided if the recent and nearby traffic information is studied.

The above problems motivate our design of an energy efficient MAC layer protocol called Convergent MAC (CMAC) which utilizes the advantages of unsynchronized sleep scheduling, anycast and unicast while mitigating the impact of their disadvantages. CMAC uses unsynchronized sleep scheduling like BMAC when there is no packet to transmit. While upon transmitting packets, CMAC first uses *aggressive RTS* (Section II-A) to *anycast* (Section II-B) packets to potential forwarders which wake up first and detect the traffic using *double channel* *check* (Section II-A). Since pure anycast based forwarding incurs higher overhead than unicast, if the sender is able to transmit packets to a node with routing metric good enough, CMAC *converges from anycast forwarding to unicast* (Section II-C). Compared to CBF, the core idea of CMAC anycast is similar. But CMAC anycast can accommodate more routing protocols, and it concentrates on activating and choosing potential forwarders instead of studying the throughput performance on mobile node scenario. Compared to GeRaF, CMAC anycast has lower overhead and more concrete radio activity detection scheme. More importantly, CMAC is implemented in TinyOS and could serve as a prototype of these ideas for future study.

To validate the practicability of CMAC, we implement CMAC in TinyOS [18] and compare it with BMAC on the Kansei testbed [19]. We also implement CMAC in *ns2* [20] and compare it with SMAC, a GeRaF variant, and 802.11 based CSMA/CA protocol. The highlights of our performance evaluation are summarized as follows:

- For static event experiments on testbed, CMAC working at 1% duty cycle significantly outperforms BMAC with the same duty cycle in throughput, latency, and energy efficiency, and exhibits similar throughput and latency performance as fully active BMAC.
- For moving event experiments on testbed, CMAC at 1% duty cycle also provides much superior performance in all aspects over BMAC with the same duty cycle.
- For moving events in simulations, CMAC achieves 95% throughput of IEEE 802.11 based unicast with comparable latency while saving up to 88.5% energy. CMAC also exhibits remarkable advantage in all aspects over SMAC and the GeRaF variant.

The rest of the paper is organized as follows. Section II details the design of CMAC, Section III presents our implementation and results gathered from Kansei testbed [19] for CMAC and BMAC, Section IV presents simulation results comparing CMAC with SMAC, IEEE 802.11 based CSMA/CA and the GeRaF variant, and Section V finally concludes the paper.

II. Convergent MAC (CMAC)

Motivated by the limitations of current approaches, we propose a new MAC layer protocol called **Convergent MAC (CMAC)** supporting low latency and high throughput as well as low duty cycle operation. CMAC has three main components, *Aggressive RTS* equipped with *double channel check* for channel assessment, *anycast* for fast forwarder discovery, and *convergent packet forwarding* for reducing the anycast overhead, and they are detailed as follows.

A. Aggressive RTS

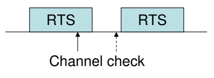
The long preamble mechanism of BMAC may incur high latency as the transmitter must ensure the receiver will be woken up before sending any data. However, the receiver may wake up much earlier than the end of the preamble, which makes part of the preamble transmission wasteful. Hence, we propose to use *aggressive RTS* to replace long preamble, which breaks up the long preamble into multiple RTS packets (also called an *(aggressive) RTS burst*). The RTS packets do not use long preambles, and are separated by fixed short gaps each of which allows receivers to send back CTS packets. The gap does not have to accommodate an entire CTS transmission as long as the RTS sender can receive the preamble of the CTS and cancel the next RTS transmission accordingly. Once the transmitter receives a CTS, it sends the data packet immediately. In this way, not only the packet delivery speeds up, but also the unnecessary energy waste on the remaining part of the preamble after the receiver has woken up is saved.

The number of RTS packets to be sent in one full aggressive RTS burst depends on the duty cycle length. For the same duty cycle length, the duration of one full RTS burst is roughly the same as the preamble of BMAC to guarantee the wake-up of the receiver. However, compared to BMAC, CMAC provides the chance for nodes to communicate as soon as possible since receivers may wake up early during a RTS burst, and hence the expected latency incurred by long preamble at each hop could be brought down by half.

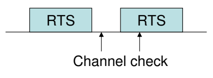
Aggressive RTS relies on receivers to detect its existence. But unlike GeRaF which can detect an RTS only if the RTS transmission starts during its awake period, CMAC uses intermittent channel check to assess the channel, and it keeps awake if the channel is busy to see whether there is any RTS intended for it. Compared to the scheme in GeRaF, CMAC can reduce the length of awake period and thus brings much lower duty cycle especially networks for rare event detection.

The low power listening (LPL) of BMAC [1] provides a energy-efficient way to quickly assess the channel, so it can serve as the basic building block of channel check in CMAC. However, the LPL is not directly applicable for aggressive RTS detection. In BMAC, the LPL is reliable since the channel will be *always* busy during any preamble transmission. But in CMAC, a gap exists between two consecutive RTS packets, and thus the channel assessment will conclude an idle channel if it happens during such a gap (given no other ongoing transmission). To resolve this problem, we propose to use *double channel check* in CMAC, which works by assessing the channel *twice* instead of once after each long sleep period. Each check takes up to five RSSI samples just like BMAC, but there is a fixed short interval between the two checks during which the radio is kept off. This interval is much shorter than the duty cycle length but slightly longer than the fixed gap between two consecutive RTS packets. If the first check detects a busy channel, the second check will be canceled (Fig. 1(a)). Otherwise, the second check is performed (Fig. 1(b)). The positive conclusion on busy channel from either check will keep the node awake anticipating an RTS. To prevent the scenario in Fig. 1(c), the small interval in one double-check must be shorter than the RTS transmission time. This can be satisfied by padding RTS packet with extra bytes. (We present how these parameters of interval lengths and packet sizes are chosen in Section III.) Such a “double-check” mechanism ensures that nodes will not miss any nearby RTS burst.

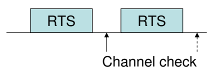
Note that the length of the interval within one double-



(a) Case 1



(b) Case 2



(c) Case 3

Fig. 1. Double Channel Check used by CMAC. In case 1, the first channel check detects a busy channel, and thus the second check (dashed arrow) is cancelled. In case 2, the first channel check happens between two RTS transmissions, but the second check will detect the busy channel. Case 3 is impossible since the interval in a double check is shorter than RTS length.

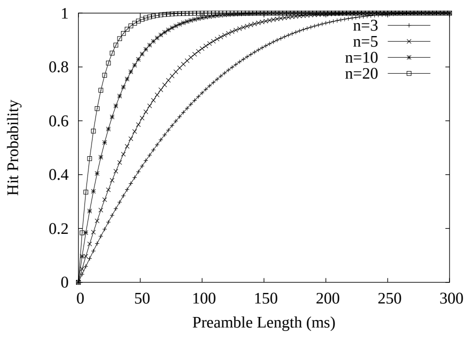


Fig. 2. The relationship between the aggressive RTS probing duration and the probability of waking up at least one potential forwarders. ($L = 300ms$)

check is fixed, and thus the gaps between consecutive RTS transmissions must also be fixed. CMAC achieves this by sending all RTS packets without assessing the channel except the first one. This may cause two RTS bursts from two transmitters to collide. Such collisions could be resolved either by one of them receiving the RTS preamble from another during its RTS sending gaps, or by the initial backoffs before sending their first RTS packets. For the former case, the interrupted RTS burst will be restarted later just like resending a long preamble, while the resolution for second case is the same as CSMA based BMAC. If an RTS transmitter detects a valid preamble before sending its subsequent RTS packets but fails to receive a CTS packet after that, it will retry the entire RTS burst with initial backoff. This could happen if another node with packets to send wakes up between two consecutive RTS packets and chooses a small initial backoff.

B. Anycast Based Forwarding

If only unicast is used, the expected latency of using aggressive RTS is roughly half that of using long preamble. However, usually there are more than one potential forwarding node within the communication range of the transmitter. They form a *forwarding set* with better but not necessarily the best routing metrics, and some of them may wake up much earlier than the one with the best metric. Thus if the data packet is *anycast* to such nodes, some progress could be made towards the destination even if the intended receiver is sleeping, and the end-to-end latency could be further reduced. Simple calculation shows for duty cycle length L and forwarding set size n, it takes on average $\frac{L}{n+1}$ to get contacted with at least one of them. For $L = 300ms$ and different n, Fig. 2 shows the relationship between the aggressive RTS probing duration and the probability of waking up at least one potential forwarder.

However, it is possible that more than one potential forwarder try to reply the same RTS. Hence a contention resolution scheme is needed for CTS transmissions. In addition, among all potential forwarders, the transmitter should favor the node with the best routing metric among the awake ones. To achieve these two objectives, we design a CTS contention resolution scheme similar as but more general than the Contention Based Forwarding (CBF) [10]–[17], and this scheme also has lower overhead than these in [6]–[9].

Similar as CBF and GeRaF, CMAC also uses routing metric to prioritizing CTS transmissions. But besides the geographical distance used by CBF and GeRaF, CMAC could use many other routing metrics, such as hop count, ETX [21], ETT [22] and PRR×Dist [23]. Generally, CMAC assumes each node knows the costs (routing metrics to the destination) of choosing different nodes in its forwarding set as next hop forwarders, and such information is also shared by nodes in its forwarding set. In TinyOS [18], these costs could be fed by the routing module easily using interfaces. Based on the costs, CMAC generates m regions $R_1 = [0, cost_1], R_2 = (cost_1, cost_2], \ldots, R_m = (cost_{m-1}, \infty]$. These regions could be evenly divided or decided according to the cost distribution. Note that the cost of choosing any neighbor node as next hop falls into one of these regions, but only nodes in the forwarding set will try to send CTS. The information of the m regions along with the routing metric of the transmitter could be carried in each aggressive RTS packet. We find that a small m like 3 provides good performance in experiments and simulations, and thus the piggybacking does not incur high overhead. Note that CMAC works the same as CBF when geographical distance is used as the routing metric, so routing beacons and extra information carrying in RTS could be saved in this case.

CMAC partitions the gap between two consecutive RTS packets into a few sub-intervals called *CTS slots*. CTS slots have one-on-one correspondence with the cost regions generated by the transmitter where regions having smaller costs are mapped to earlier CTS slots. After obtaining the cost regions in an aggressive RTS packet, each potential forwarder calculates which region it belongs to, and tries to send CTS in the corresponding CTS slot. Each CTS slot is further divided into several *mini-slots* to resolve the contention within each region, and each receiver will randomly choose one mini-slot to start its CTS transmission (Fig. 3). On detecting busy channel before transmitting CTS, pending CTS transmission will be canceled assuming the existence of another CTS.

We use geographical distance as routing metric to illustrate the anycast route selection of CMAC. The area that is closer to the sink than the sender and within its transmission range is divided into m regions, $R_1, R_2, \ldots, R_m$, such that all nodes in R_i are closer to the sink than nodes in R_j for $i < j$. Fig. 4 shows an example where $m = 3$. Then each node in region R_i schedules its CTS transmission in a randomly chosen mini-slot of CTS slot i (Fig. 3). If the packet forwarding encounters a "void" where the forwarding set is empty, the transmitter could simply use "right hand rule" with the ID of the receiver

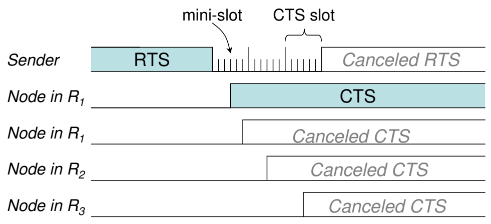


Fig. 3. CTS contention resolution. Nodes in different routing metric regions transmit their CTS in the corresponding CTS slots, while nodes in the same region transmit their CTS packets in randomly selected mini-slots. The first sent CTS will cancel other CTS and subsequent RTS transmissions.

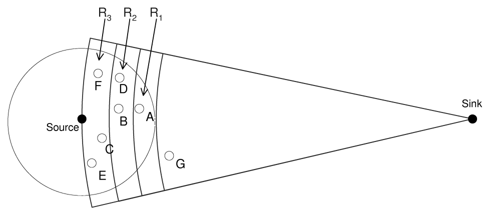


Fig. 4. *Example of cost region generation in CMAC using geographical distance as routing metric.*

specified in destination field of RTS packets. Sophisticated void circumvention is out of the scope of this paper.

C. Converging from Anycast to Unicast

Although anycast obviates the need for synchronization messages and has better chance to make progress in packet forwardings than unicast, it has two main shortcomings. First, anycast may choose suboptimal routes because the best next hop is sleeping or due to interference. Second, the overhead of anycast RTS/CTS exchange is usually higher than its unicast counterpart. Hence, a mechanism is needed to reduce the overhead incurred by anycast, and we propose the *convergent packet forwarding* to resolve these problems as follows.

In CMAC, the node will remain awake for a short duration after receiving a data packet from anycast. During this period, a node with better routing metric could wake up and become the receiver of the next anycast. If the latest anycast receiver has a routing metric close to the best, CMAC will use unicast instead to avoid anycast overhead. As an example, the route going through any node in region R_1 (Fig. 4) is close to the optimal route. However, it is possible that nodes with good enough routing metrics may not exist. For example, this will happen if there is no node in region R_1 (Fig. 4). Hence if the transmitter cannot find a better next hop than current one after a duty cycle length, it starts to unicast packets to current receiver and updates its cost regions. In this way, the packet forwarding progressively converges from anycast to unicast as the example shown in Fig. 5. After some time without successful data packet reception, CMAC will timeout and nodes will follow unsynchronized idle duty cycles.

The unicast after such convergence may or may not use normal RTS/CTS. In our experiments, CMAC does not use RTS/CTS after convergence for the comparison with BMAC. In our simulations, CMAC uses normal RTS/CTS that is similar as 802.11 after convergence for the comparison with

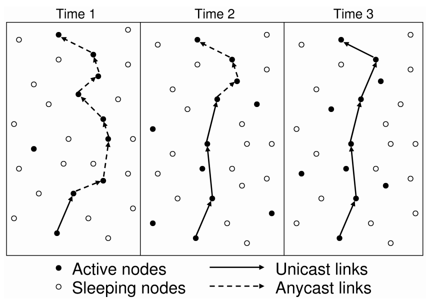


Fig. 5. *Anycasting route converges to unicast route gradually.*

802.11, SMAC and GeRaF.

If the event moves fast, the source nodes may continuously change with each of them generating only a small number of packets. In this case, the convergence may still happen after the merge points of different flows. For some other cases such as low data rates, the convergence may not happen, but CMAC can still use aggressive RTS and anycast to make quick progress towards the sink.

D. Synchronized Wake-up Schedule

In order to save more energy after convergence, nodes can synchronize with their neighbor nodes to use some kind of wake-up schedule instead of keeping fully awake. In our simulations, we evaluate a CMAC variant using a staggered scheduling similar to DMAC [4] after convergence. When the transmitter intends to converge from anycast to unicast, it synchronizes its schedule with the receiver. The two nodes will maintain the staggered schedule as long as there is traffic between them. After a certain duration without traffic, the nodes go back to use unsynchronized duty cycling. This CMAC variant are detailed in our technical report [24].

III. Experimental Evaluation

We implement[1] CMAC in TinyOS [18], and compare CMAC with BMAC[2] on the Kansei testbed [19]. This section first describes our implementation and the experiment methodology, and then presents the results from the testbed.

A. Implementation and Experiment Methodology

Our implementation is based on XSM [25] which is similar as Mica2 mote [26] in CC1000 radio [27] and processing board. We set the mini-slot length to the transmission time of 1 byte on CC1000 radio which is $416\mu s$, a period long enough to accommodate the propagation delay and busy channel detection (One channel sampling takes about 265μ to finish). One CTS slot is set to 6 mini-slots, and the number of CTS slots is set to $m = 3$. Thus the gap between two consecutive RTS packets has to accommodate the transmission of 18 bytes which is roughly $8ms$. Then $10ms (> 8ms)$ is chosen as the interval in one double-check. The aggressive RTS packet size is set to 44 bytes including preamble and padding bytes such that an RTS transmission duration is longer than $10ms$. The

[1]Code available at `http://www.cse.ohio-state.edu/~liusha/cmac`.
[2]We plan to compare CMAC with BMAC plus SP in the future due to the current unavailability of SP code in TinyOS distribution.

aggressive RTS packet is comparable in size with the data packet in TinyOS (36 bytes), but aggressive RTS serves mainly to contact the receiver instead of providing collision avoidance, and it is still much shorter than long preambles. Take duty cycle length of $300ms$ as an example, BMAC uses 744-byte preamble which is much longer than one 44-byte RTS. The padding bytes in each RTS after convergence could be saved if the RTS/CTS based collision avoidance is desired.

The Kansei testbed [19] consists of 105 XSM nodes forming a 15×7 topology with node separation of 3 feet. The transmission range is set to 4 rows/columns in the testbed. Each XSM node is attached to a Linux-based stargate [28], and those stargates are connected using Ethernet to the Kansei server. Instead of creating real events, we issue messages to XSM nodes as commands to emulate an event. Such command messages are broadcast from the Kansei server and forwarded by serial forwarder programs on stargates to the UART ports of XSM nodes. XSM nodes then decide whether they "detect" the event based on the prescribed event information.

We evaluate the performance of CMAC for event-triggered sensor networks, in which the packets are generated only when sensors observe an event. We evaluate the throughput, latency and energy efficiency of CMAC against BMAC for two basic event scenarios, static event and moving event. Throughput refers to the total number of packets received at the sink in 600 seconds, latency is the average delay endured for a packet to reach the sink, and energy efficiency refers to the energy consumption of the entire network for delivering one 36-byte packet to the sink, and is called normalized energy. To measure latency, nodes record the start time after receiving the initiation command message. The time difference between the packet generation and the recorded start time, denoted by t_1, is carried in that packet. After receiving this packet, the sink calculates the difference between the reception time and its own recorded start time, denoted by t_2. Then $t_2 - t_1$ is used as the latency of this packet. (We ignore the variation in delay for the command message to reach each XSM node as it is insignificant compared to the packet delivery latency.) To calculate the energy consumption, we measure the time each node spends on transmitting, receiving and channel checking, and the energy consumption is calculated using the current consumption values provided in Table 2 of [1].

Note that the double channel check almost doubles the times of channel sampling in BMAC. Thus theoretically CMAC consumes more energy on channel assessment than BMAC using the same duty cycle length. To be fair, we evaluate CMAC with duty cycle length double that of BMAC in this section. For example, if BMAC uses $300ms$ duty cycle length, CMAC will use $600ms$. Since using $300ms$ duty cycle length in BMAC is roughly 1% duty cycle, we denote it by BMAC 1%, and denote CMAC using $600ms$ duty cycle length as CMAC 1%. To provide the baseline for throughput and latency evaluation, we also gathered the data for BMAC and CMAC without duty cycling, denoted by BMAC 100% and CMAC 100% respectively. The performance of CMAC 100% is provided to show CMAC working at 100% duty cycle is similar as always active BMAC.

B. *Static Event Scenarios*

In this set of experiments, we evaluate the throughput, latency and energy efficiency of CMAC and BMAC for static event scenario. We emulate an event happening at one corner of the testbed using the method described in Section III-A. The source node sends all packets to the sink located at the diagonally opposite corner. We vary the data rate at source nodes, and the results are shown in Fig. 6.

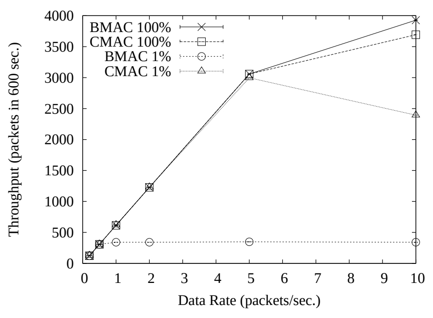
For low data rates ($0.2 \sim 0.5$ packets/sec.), both CMAC 1% and BMAC 1% can deliver all packets (Fig. 6(a)), but Fig. 6(b) shows that CMAC 1% exhibits better latency performance than BMAC 1% due to the capability of aggressive RTS and anycast to discover awake potential forwarders.

Under high data rates (≥ 1 packet per second), BMAC 1% can not deliver all packets to the sink, and the flat curve shows the channel capacity is reached (about 300 packets in 600 seconds) due to long preambles and multihop contention. CMAC 1% saves unnecessary long preambles, and thus not only significantly outperforms BMAC 1% but also provides similar throughput as BMAC 100% and CMAC 100% (Fig. 6(a)). In some cases, e.g., data rate of 2 and 5 packets per second, CMAC 1% even provides latency performance very close to that of BMAC 100% (Fig. 6(b)). This is due to the convergence of CMAC from anycast to unicast and the saving on anycast overhead. At the data rate of 10 packets per second, CMAC 1% does not provide throughput and latency very close to BMAC 100% or CMAC 100% because the high contention leads to some convergence duration timeouts which result in more RTS/CTS, but CMAC 1% still exhibits significant improvement over BMAC 1%.

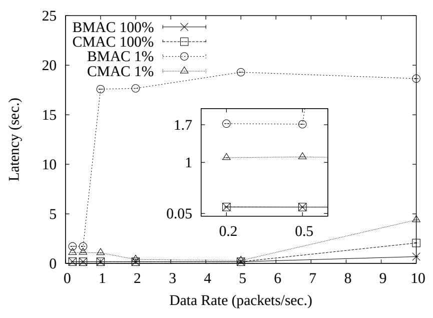
Fig. 6(c) shows CMAC 1% utilizes the energy more efficiently than BMAC 1% and BMAC 100%, and the energy efficiency becomes better as the data rate increases. It can also be observed that the energy efficiency of BMAC 1% is even worse than BMAC 100% for data rates beyond 5 packets per second because the preamble length for 1% duty cycle is not efficient in these cases. Hence, we conclude that CMAC is more suitable for providing high throughput and low latency while the idle duty cycle is low.

C. *Moving Event Scenario*

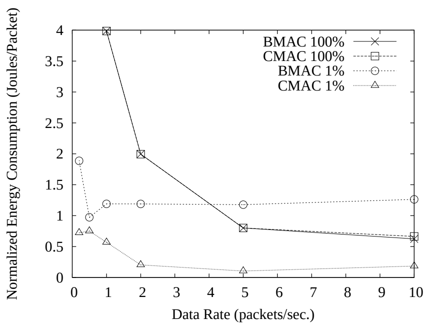
To evaluate the performance of CMAC when the event is moving, we move the emulated event along the bottom edge of the testbed at different speeds (Fig. 7). A program on the Kansei server calculates when and which nodes are supposed to detect the event, and it triggers packet generations at these time points by sending messages to these nodes. Each node generates one packet once it receives such a message. The event restarts from the bottom left corner each time it reaches the bottom right corner. The sink is located at the top right corner of the testbed, and each experiment duration is 600 seconds. We vary the speed of event moving such that faster movement will trigger more packets. The results of throughput, latency and energy efficiency are shown in Fig. 8.



(a) Throughput



(b) Latency



(c) Energy Efficiency

Fig. 6. Experiment results of throughput, latency and energy efficiency performance of CMAC and BMAC under different data rates.

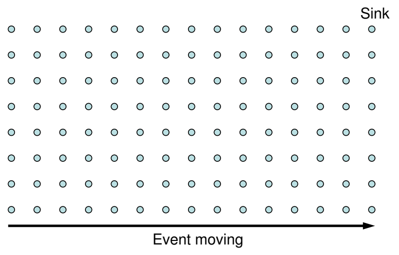


Fig. 7. Moving event scenario in experimental evaluation. The sink locates at top right corner, and the event moves along the bottom edge at different speeds

Fig. 8(a) shows the advantage of CMAC 1% over BMAC 1% in throughput. The throughput of BMAC 1% increases with the increase of the moving speed for slow speeds, but it gradually drops after the moving speed exceeds 1 row/sec. However, the throughput of CMAC 1% increases proportionally with the increase of moving speed with 100% packet delivery ratio. Fig. 8(b) shows remarkable advantage of CMAC 1% over BMAC 1% in latency. Unlike BMAC 1% whose latencies are in tens of or even over one hundred seconds, CMAC 1% provides less than one second latency on average to deliver one packet to the sink. For BMAC 1%, the queueing delay contributes most of the latency and is caused by using long preamble in each packet transmission. While for CMAC 1%, due to its capability to find a “quick” forwarder in the forwarding set and the convergence from anycast to unicast, heavy queuing is avoided and the latency is much lower. Fig. 8(c) shows the advantage of CMAC 1% in energy efficiency. CMAC 1% saves $75\% \sim 95\%$ energy of BMAC 1% to deliver one packet. In addition, the normalized energy consumption of CMAC 1% decreases gradually with the increase of moving speed. This is because CMAC has better chance to converge to unicast when the event moves faster, where more packets are generated and the convergence happens after the merge points of multiple flows. But for BMAC 1%, the energy efficiency increases sharply due to the inefficiency of long preamble for fast event moving speeds.

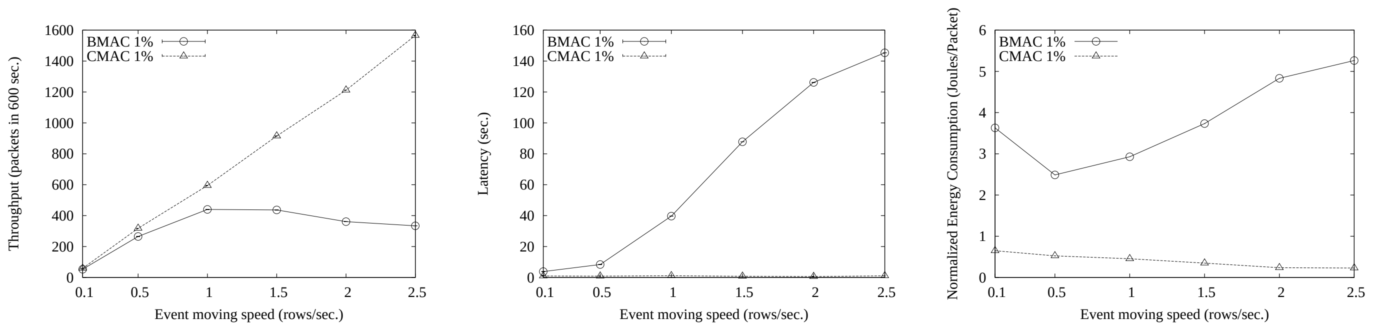
D. Anycast Performance

For low data rates, CMAC may not be able to converge from anycast to unicast since the awake period after receiving a data packet may timeout before the next packet arrives. In such cases, the performance of CMAC depends on the aggressive RTS and anycast mechanism. Thus we evaluate the performance of the aggressive RTS and anycast mechanism in this section. The duty cycles are 1% and 0.1%, where each cycle is $3000ms$ and $6000ms$ respectively for BMAC 0.1% and CMAC 0.1%. The source node is located at one corner, and the sink is at the diagonally opposite corner. We vary the node density by adjusting the transmission range from 3 rows/columns to 8 rows/columns and run each experiment for 600 seconds. The data rate is chosen such that every packet is purely anycast enroute without convergence to unicast or experiencing queuing delay. Thus for each generated packet, it is a fresh start for the entire network.

Since the Kansei testbed has fixed physical topology, increasing transmission range leads to decrease in hop count. Therefore, we present the latency normalized by the hop count of unicast, i.e., $Latency/Hops$, and the results are shown in Fig. 9(a) and 9(b) (Figures for throughput are omitted since all protocols can deliver all packets to the sink).

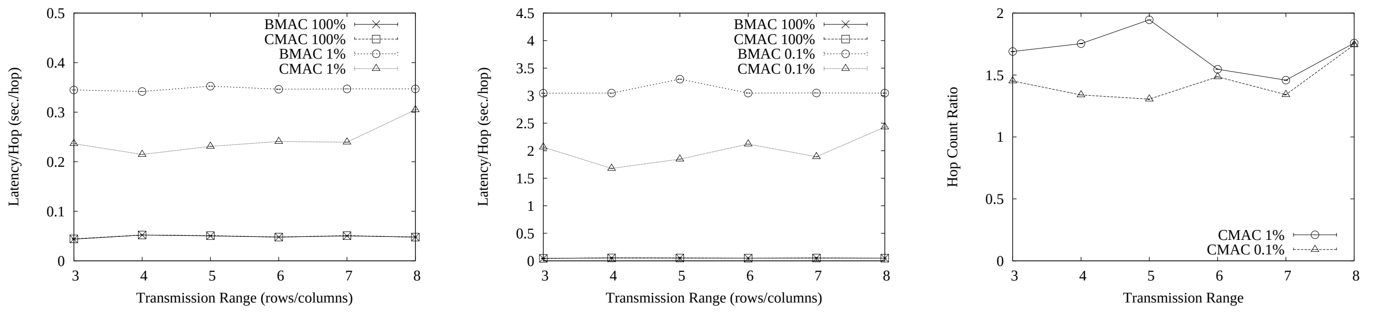
CMAC reduces the latency of BMAC by about 33% at both 1% and 0.1% duty cycles except for transmission range of 8 rows with 1% duty cycle, where the improvement is not very significant. The reason for this is that there are only two hops for unicast if one transmission can go through 8 rows of the testbed, and the optimization space left for CMAC is only one hop since CMAC does not use anycast once the sink can be reached in one hop. The per hop latency does not decrease with the increase of node density in Fig. 9(a) and 9(b), this is because anycast may have a small forwarding set near the sink due to the limited scale of the Kansei testbed.

We also collect the route stretch of anycast, which is represented by the average number of hops of anycast normalized by the hop count of unicast. Fig. 9(c) shows CMAC 0.1% has larger stretch than 1%. This is because for higher duty cycle, the probability for more than one node to wake up and reply the same RTS is also higher, and better routing metrics could be found among more awake nodes. As Fig. 9(a) and 9(b) show, even with route stretch, CMAC 1% can still outperform BMAC 1% because of the fast progress made to the destination using aggressive RTS and anycast.



(a) Throughput (b) Latency (c) Energy Efficiency

Fig. 8. Experiment results of throughput, latency and energy efficiency performance of CMAC and BMAC under different event moving speeds.



(a) Per Hop Latency for 1% Duty Cycle (b) Per Hop Latency for 0.1% Duty Cycle (c) Route Stretch

Fig. 9. Experiment results of anycast latency performance of CMAC 1% and CMAC 0.1% under different node densities.

IV. Simulation Based Evaluation

We also conduct simulations[3] for large scenarios to compare the throughput, latency and normalized energy consumption of CMAC with other protocols using the network simulator *ns2* [20]. Our study is based on the following six protocols:

- **CSMA/CA:** Using unicast RTS-CTS-DATA-ACK 802.11 protocol with radio fully awake. This protocol servers as the baseline for throughput and latency performance.
- **Anycast:** Using the anycast mechanism described in Section II-B with radio fully awake.
- **GeRaF:** Using the anycast protocol in Section II-B with 10% duty cycle and $3ms$ active period. This GeRaF variant does not use busy tone, and the anycast protocol is similar in essence to but slight different from [6] [7].
- **CMAC:** Our proposed scheme described in Section II working on 1% idle duty cycle.
- **CMAC-S [24]:** Similar to CMAC, but after convergence, nodes use a DMAC-like [4] staggered wake-up schedule.
- **SMAC:** SMAC with adaptive listening working at 10% duty cycle. In our simulation, SMAC can barely send packets to the sink using 1% duty cycle, therefore we use 10% duty cycle for SMAC.

The simulations are conducted on $2000m \times 2000m$ network with an event moving randomly at $10m/s$. We use $250m$ as the transmission range, but our protocol works for any radio transmission range. Other parameters are shown in Table I.

[3]Code available at `http://www.cse.ohio-state.edu/~liusha/cmac`.

TABLE I
SIMULATION PARAMETERS

Tx range	$250m$	RTS size	14 bytes
Bandwidth	$38.4Kbps$	CTS size	14 bytes
Tx power	$27mA$	ACK size	28 bytes
Rx power	$10mA$	Data header	20 bytes
Idle power	$10mA$	Data payload	50 bytes
CTS slot	$0.2ms$	Anycast CTS	22 bytes
Active period	$3ms$	Preamble+PLCP	24 bytes

In this section, we present results in mobile event scenarios for varying initial idle duty cycle, node density, and data rate. More simulation results on other aspects, such as static event and link quality, are available in our technical report [24].

A. Initial Duty Cycle

First we evaluate the impact of the idle duty cycle by varying it from 0.1% to 1%. This set of tests shows which protocol has more potential to work at low duty cycles. Fig. 10 shows the performance for different idle duty cycles using data rate of 10 packets/s. In the simulations, GeRaF and SMAC can barely deliver any packet to the sink when working under 1% duty cycle. Therefore we use 1% to 10% for them.

We can see that the throughput decreases gradually with the decrease of initial duty cycle. For CMAC, the throughput decreases about 25% when duty cycle changes from 1% to 0.1%, and the normalized energy consumption decreases about 30%. However, this energy saving at 0.1% duty cycle comes at the expense of higher initial detection latency. Through

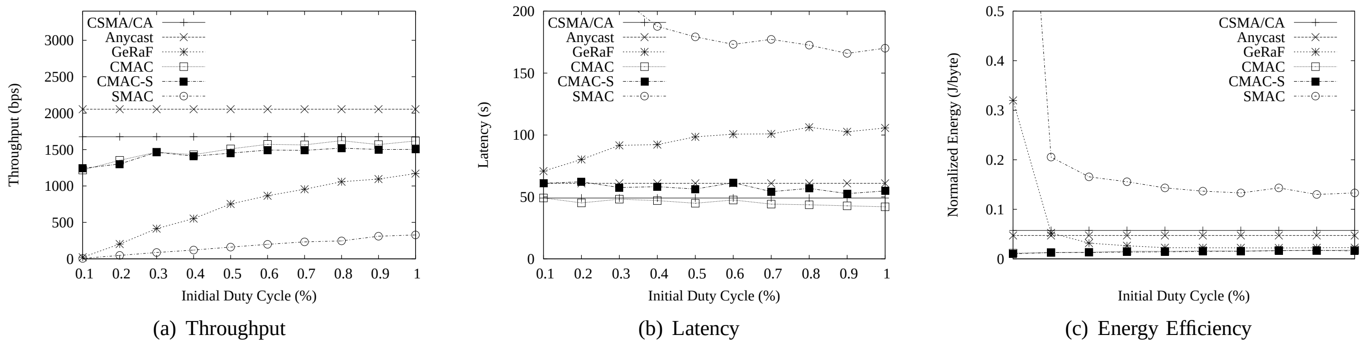


Fig. 10. Simulation results for throughput, latency and energy efficiency performance of CMAC, SMAC, GeRaF and CSMA/CA under different idle duty cycles. In our simulations, GeRaF and SMAC barely receive packets at the sink when working below 1% duty cycle, therefore we use 1% to 10% duty cycle for GeRaF and SMAC in the simulation. SMAC has high delay when the duty cycle is less than 3% and is not shown in the figure.

tracing the simulation data, we found that the delay for the first packet to arrive at the sink increases from 2 seconds to near 14 seconds, but this is still better than SMAC and GeRaF since they can not deliver even one packet to the sink with such a low duty cycle (0.1%).

CMAC and CMAC-S have lower throughput compared to CSMA/CA and Anycast protocols due to several reasons. First, the duty cycle is only 1% initially, which limits the initial throughput. Second, the event may move out of the sensing range of a source node before the convergence fully completes. However, CSMA/CA and Anycast achieve higher throughput as the cost of very high power consumption as nodes remain 100% awake. Anycast has the highest throughput because of its ability to forward packets through multiple paths, but this high throughput is achieved by keeping all nodes active all the time which is not energy efficient.

SMAC with 10% duty cycle can only achieve $1/5$ throughput of CMAC 1%, together with latency 4 times and normalized energy consumption 8 times that of CMAC 1%. In SMAC, transmissions can only happen during active periods. Thus the available time for transmissions is quite limited.

The throughput of GeRaF is lower than CMAC even if it works on the duty cycles 10 times higher due to two main reasons. First, GeRaF uses anycast for each packet which incurs higher overhead. Second, in GeRaF, nodes receive RTS only if they happen to wake up during the preamble transmission of the RTS, which is inefficient compared to pulse based double channel check in CMAC.

B. Node Density

Next we evaluate the performance of CMAC in networks with different node densities. We vary the number of nodes in the network from 100 to 625 while keeping the area and event size unchanged.

From Fig. 11 we can see that the throughput, latency and normalized energy consumption all increase with increase of node density. This is because more nodes are generating packets with higher node density. The throughput of anycast is the best because it can always take alternate path during high contention. CMAC provides similar throughput as CSMA/CA and outperforms others (Fig. 11(a)). For latency, CMAC is also among the best. More importantly, CMAC outperforms all other protocols in normalized energy consumption.

C. Data Rate

Fig. 12 shows the simulation results of throughput, latency and normalized energy consumption of different protocols for different date reporting rates. CMAC and CMAC-S use the least energy, while achieving about 95% of throughput of CSMA/CA (at data rate 10 pkts/s).

V. Conclusion

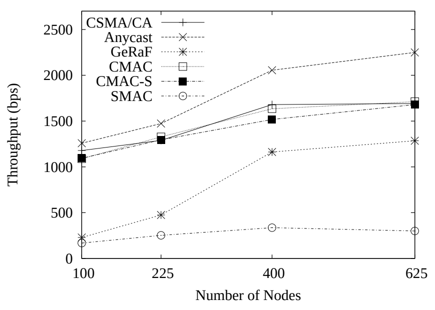
Existing MAC layer solutions for low duty cycling either consume a lot of energy on periodic synchronization messages or incur high latency due to the lack of synchronization. Thus in this paper we proposes CMAC, a MAC layer protocol for maximizing network lifetime while maintaining high throughput and low latency for sensor networks. During idle periods, nodes follow unsynchronized low duty cycles and use a novel technique called *double channel check* to assess the channel. When transmitting packets, CMAC initially uses aggressive RTS and anycast to exploit the diversity in the forwarding set, and then converges to unicast to reduce the overhead.

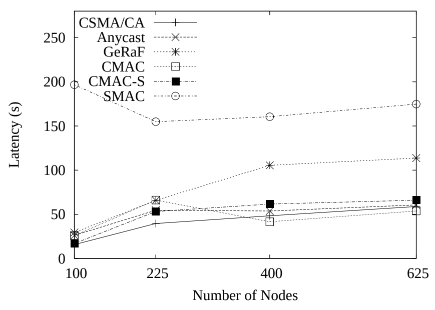
The experiment results from Kansei testbed show that CMAC 1% can approach the throughput and latency performance of fully awake BMAC, while outperforming BMAC 1% in all aspects. Using simulations, we compare CMAC with SMAC, CSMA/CA and a GeRaF variant in moving event scenarios. CMAC not only outperforms SMAC and the GeRaF variant, but also achieves 95% throughput of CSMA/CA with similar latency and 88.5% less energy. With higher tolerance for initial latency, CMAC can even work at 0.1% duty cycle with long-term throughput comparable to CSMA/CA.

Based on our study, we conclude that CMAC is highly suitable for wireless sensor networks that require low latency and high throughput as well as long network lifetime.

References

[1] J. Polastre, J. Hill, and D. Culler, “Versatile Low Power Media Access for Wireless Sensor Networks,” in *Proc. SENSYS’04*, Nov. 2004, pp. 95–107.


(a) Throughput


(b) Latency

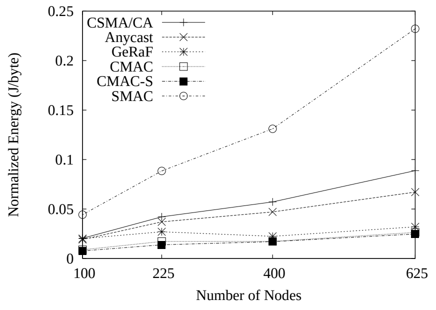
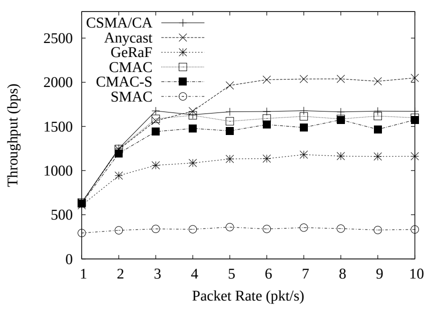
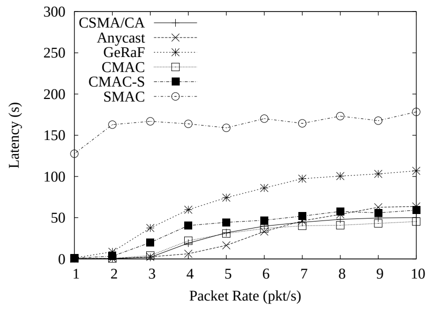

(c) Energy Efficiency

Fig. 11. Simulation results for throughput, latency and energy efficiency performance of CMAC, SMAC, GeRaF and CSMA/CA under different node densities.


(a) Throughput


(b) Latency

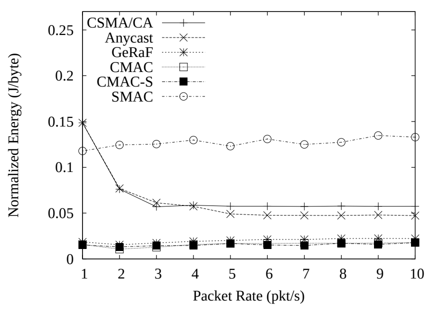

(c) Energy Efficiency

Fig. 12. Simulation results for throughput, latency and energy efficiency performance of CMAC, SMAC, GeRaF and CSMA/CA under different data rates. CMAC and CMAC-S can achieve 90% to 95% throughput of CSMA/CA.

[2] W. Ye, J. Heidemann, and D. Estrin, "Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks," *IEEE/ACM Trans. Networking*, vol. 12, no. 3, pp. 493–506, June 2004.

[3] T. van Dam and K. Langendoen, "An Adaptive Energy-Efficient MAC Procotol for Wireless Sensor Networks," in *Proc. SenSys'03*, Nov. 2003, pp. 171–180.

[4] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An Adaptive Energy-Efficient and Low-Latency MAC for Data Gathering in Wireless Sensor Networks," in *Proc. IPDPS'04*, Apr. 2004, pp. 224–231.

[5] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A Unifying Link Abstraction for Wireless Sensor Networks," in *Proc. SenSys'05*, Nov. 2005, pp. 76–89.

[6] M. Zorzi and R. R. Rao, "Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Multihop Performance," *IEEE Trans. Mobile Comput.*, vol. 2, no. 4, pp. 337–348, Oct. 2003.

[7] ——, "Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Energy and Latency Performance," *IEEE Trans. Mobile Comput.*, vol. 2, no. 4, pp. 349–365, Oct. 2003.

[8] S. Jain and S. R. Das, "Exploiting Path Diversity in the Link Layer in Wireless Ad Hoc Networks," in *Proc. WoWMoM'05*, June 2005, pp. 22–30.

[9] R. R. Choudhury and N. H. Vaidya, "MAC-Layer Anycasting in Ad Hoc Networks," *SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 75–80, Jan. 2004.

[10] H. Füßler, J. Widmer, M. Käsemann, M. Mauve, and H. Hartenstein, "Contention-Based Forwarding for Mobile Ad Hoc Networks," *Ad Hoc Networks*, vol. 1, no. 4, pp. 351–369, Nov. 2003.

[11] B. Blum, T. He, S. Son, and J. Stankovic, "IGF: A State-Free Robust Communication Protocol for Wireless Sensor Networks," Technical Report CS-2003-11, 2003. [Online]. Available: http://www.cs.virginia.edu/∼techrep/CS-2003-11.pdf

[12] M. Heissenbüttel, T. Braun, T. Bernoulli, and M. Wälchli, "BLR: Beacon-Less Routing Algorithm for Mobile Ad-Hoc Networks," *Computer Communications*, vol. 27, no. 11, pp. 1076–1086, July 2004.

[13] P. Škraba, H. Aghajan, and A. Bahai, "Distributed Passive Routing Decisions in Mobile Ad-Hoc Networks," in *Proc. VTC'05*, vol. 4, Sept. 2004, pp. 2814–2818.

[14] D. Chen, J. Deng, and P. K. Varshney, "A State-Free Data Delivery Protocol for Multihop Wireless Sensor Networks," in *Proc. WCNC'05*, vol. 3, Mar. 2005, pp. 1818–1823.

[15] D. Chen, G. Cao, and L. Zuo, "A Multihop Data Relay Scheme for Wireless Networked Sensors," in *Proc. VTC'05*, vol. 3, Sept. 2005, pp. 1814–1818.

[16] D. Chen, J. Deng, and P. K. Varshney, "On the Forwarding Area of Contention-Based Geographic Forwarding for Ad Hoc and Sensor Networks," in *Proc. SECON'05*, Sept. 2005, pp. 130–141.

[17] M. Witt and V. Turau, "BGR: Blind Geographic Routing for Sensor Networks," in *Proc. WISES'05*, May 2005.

[18] "TinyOS," http://www.tinyos.net.

[19] "Kansei testbed," https://cast.cse.ohio-state.edu/kansei/.

[20] "The Network Simulator – ns-2," http://www.isi.edu/nsnam/ns/.

[21] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *Proc. MobiCom'03*, Sept. 2003, pp. 134–146.

[22] J. Padhye, R. Draves, and B. Zill, "Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks," in *Proc. MobiCom'04*, Sept.

[23] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari, "Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks," in *Proc. SenSys'04*, Nov. 2004, pp. 108–121.

[24] K.-W. Fan, S. Liu, and P. Sinha, "Convergent Anycast: A Low Duty-Cycle MAC Layer for Sensor Networks," Technical Report OSU-CISRC-4/05–TR24.

[25] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events," in *Proc. IPSN'05*, Apr. 2005, pp. 497–502.

[26] "Mica2," http://www.xbow.com/Products/productsdetails.aspx?sid=72.

[27] "CC1000," http://www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf.

[28] "Stargate," http://platformx.sourceforge.net/home.html.