9-30-2009

Power Efficiency of Cooperative Communication in Wireless Sensor Networks

Sunita Gupta
University of Nebraska-Lincoln, sunita.s.gupta@gmail.com

Mehmet C. Vuran
University of Nebraska-Lincoln, mcvuran@cse.unl.edu

M. Cenk Gursoy
University of Nebraska-Lincoln, gursoy@engr.unl.edu

3rd International Conference on Signal Processing and Communication Systems
Omaha, Nebraska, 28-30 September 2009

Power Efficiency of Cooperative Communication in Wireless Sensor Networks

Sunita Gupta
Electrical Engineering
University of Nebraska—Lincoln
sunita.s.gupta@gmail.com

Mehmet C. Vuran
Computer Science and Engineering
University of Nebraska—Lincoln
mcvuran@cse.unl.edu

M. Cenk Gursoy
Electrical Engineering
University of Nebraska—Lincoln
gursoy@engr.unl.edu

***Abstract*—In this paper, a new protocol termed CoopXLM that integrates cooperative communication and Cross-Layer Module (XLM) is created and examined via simulation. CoopXLM modifies XLM by allowing multiple cooperative nodes to participate in receiver-based contention. Although there is increased energy consumption for processing, simulation results indicate that across all duty cycles there is an average energy savings of 38% with CoopXLM in comparison to XLM. It was also found that CoopXLM yields higher goodput than XLM at lower duty cycles but yields lower goodput at higher duty cycles.**

I. Introduction

Wireless communication systems have recently gained popularity as their benefits are being acknowledged and engineering ingenuity continues to overcome their inherent challenges. One serious disadvantage of wireless sensor networks (WSNs) is that the sensors used in these networks are often limited to the use of a single battery, and their success is highly dependent upon power efficient protocols. Traditional layered protocols often prove to be inefficient for WSNs, and determining whether more power efficient protocols can be developed is important to the future advancement of these networks. One recently developed protocol, the cross-layer module (XLM) [1], has been shown to increase network efficiency and reliability in comparison to traditional layered protocols. XLM melts together the physical, MAC, network and transport layers. At the core of XLM is initiative determination, which is detailed in [1].

Cooperative communications is also an alternative to traditional protocols that can increase power efficiency in WSNs by having several nodes simultaneously transmit a single message to the intended destination. The result is an energy savings due to the wireless broadcast advantage (WBA). The WBA stems from the fact that when a wireless node transmits a packet, all nodes within the transmission radius are able to listen. Usually, nodes for which the packet is not intended ignore the packet, but it is possible for nodes to accept all incoming packets. When a node can communicate with a group of nodes by only transmitting once, instead of transmitting to each receiving node individually, the node is using the WBA [3].

In this paper, a protocol named CoopXLM that integrates cooperative communication and XLM is presented. CoopXLM retains the initiative determination of XLM and adds the ability for multiple nodes to participate in a single transmission. Simulation results show that CoopXLM provides energy savings when compared to XLM. However, CoopXLM only improves unique goodput when compared to XLM at lower duty cycles. At higher duty cycles, XLM has a higher unique goodput than CoopXLM.

The remainder of the paper is organized as follows. Section II describes related work. Section III begins with definitions and assumptions. Then it continues with a description of the aspects of CoopXLM that make it unique from XLM. Last, it contains a detailed description of the CoopXLM protocol. Section IV explains the simulation parameters and results. Finally, Section V presents a summary of conclusions.

II. Related Work

The Open Systems Interconnection (OSI) layers have been used extensively in wired networks to provide portability and modularity. Researchers and designers are able to focus on optimizing a certain layer while treating the remaining layers as black boxes. OSI layers work well in wired networks because these networks have limited interactions between layers. However, there are many inter-layer effects in a wireless network. Thus, although the OSI layers have led to simplicity of system integration for wired networks, they can lead to suboptimal system performance in wireless ones. Take for example the case of packet loss; the transport layer will attribute the problem to congestion in the neighboring network layer even though the problem may be caused by bursty interferences at the MAC layer. This misdiagnosis due to abstraction will cause lower throughput [2]. Empirical studies have shown that wireless channel characteristics, whose impact would traditionally be confined to the physical layer, actually affect all layers in terms of performance [9]. In addition, the medium access control (MAC) and routing layers significantly influence each other due to interference. The physical and transport layers are coupled due to the broadcast nature of wireless communication. As the power level of each node is increased, the probability of collisions between packets increases. In order to address inter-layer effects, a protocol that combines several layers needs to be developed. Some previous works have combined a few layers, such as the MAC and routing layers, or else they have combined all layers but not implemented the design. In order to jointly optimize

978-1-4244-4474-8/09/$25.00 ©2009 IEEE

several layers and remove negative inter-layer effects, a unified cross-layer protocol is required. Cross-Layer Module (XLM) is such a protocol [1].

XLM is a cross-layer protocol designed specifically for efficiency and reliability in WSN communication. The main concepts of XLM include receiver-based contention, initiative determination, initiative-based forwarding, local congestion control and distributed duty cycle operation. XLM melts the transport, network, MAC and physical layers together. Unlike proactive routing protocols, XLM does not determine the route from the source to the sink before the need arises. Instead, XLM waits until the source has data to send to the sink; therefore, it is a reactive protocol. Although the sensor networks considered in this paper are stationary, some of them have nodes with duty cycles other than 100%. Sleeping nodes are equivalent to dead nodes for a particular communication. Each live node has two duties: source duty and router duty. The source duty is only necessary when an event occurs in the node's transmission radius. In this case, the node is responsible for generating and transmitting the packet towards the sink. The router duty is the node's duty to receive and forward packets that other nodes have generated [1]. A basic assumption is that all nodes know their own location and that of the sink.

In order to explain XLM, a walk-through of a single communication is presented next. When a source node (src) wants to send data to the sink, it listens to the medium to check if other signals are being broadcast. If the medium is busy, src performs a contention window size backoff. Once the backoff timer expires, src broadcasts a request to send (RTS) to all nodes within its transmission radius. The transmission radius is denoted in Fig. 1 by the dashed circle. The sink may or may not be within the transmission radius. If it is, then the sink becomes the chosen next hop; otherwise, receiver-based contention, which is shown in Fig. 1, ensues.

Receiver-based contention requires dividing the nodes in the transmission radius into nodes that are in the feasible region and nodes in the infeasible region. The nodes that are closer to the sink than src are in the feasible region. In Fig. 1, nodes A, B and C are feasible nodes. All other nodes are considered to be in the infeasible region. In this example, nodes D and E are in the infeasible region. D and E go to sleep while A, B and C contend for the packet. The feasible nodes each perform initiative determination, which means they calculate the result of the initiative function $\mathcal{I}$:

$$\mathcal{I} = \left\{ \begin{array}{l} 1, if \left\{ \begin{array}{ll} \xi_{RTS} \geq \xi_{Th} & \text{(Received SNR)} \\ \lambda_{relay} \leq \lambda_{relay}^{Th} & \text{(Packet relay rate)} \\ \beta \leq \beta^{max} & \text{(Buffer length)} \\ E_{rem} \geq E_{rem}^{min} & \text{(Remaining energy)} \end{array} \right. \\ 0, otherwise \end{array} \right. \quad (1)$$

The conditions in the initiative function correspond to the attributes that are important to a sensor node. The first condition, $\xi_{RTS} \geq \xi_{Th}$, determines whether the feasible node and src have acceptable channel conditions for further communication by calculating the signal-to-noise ratio (SNR) of the received RTS. The next two conditions constitute

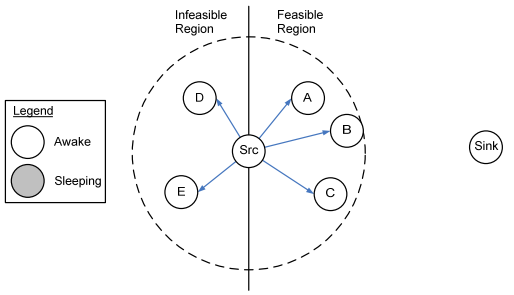

Fig. 1. Broadcasting a RTS

XLM's local congestion control. The second condition indicates that the feasible node tries to avoid local congestion by keeping the local packet relay rate, λ_{relay}, below a congestion threshold, λ_{relay}^{Th}. Since sensor nodes have limited memory and buffer overflows are also part of congestion, the third condition, $\beta \leq \beta^{max}$, checks to make sure the sensor node has enough memory available to receive another packet. Finally, in order to encourage uniform energy consumption throughout the network, the last condition states that nodes will not accept a packet if their remaining energy, E_{rem}, gets below a remaining energy threshold, E_{rem}^{min}. All nodes in the feasible region that have an initiative $\mathcal{I} = 1$ set a clear to send (CTS) backoff timer. In this example, C does not have enough remaining energy, so its initiative is 0. Thus, only A and B set their CTS backoff timers.

Fig. 2 shows the state of the nodes after the initiative function is run. The gray nodes are sleeping because they are either not in the feasible region or their initiative function is 0; thus, they will not be involved in this transmission.

The length of the CTS backoff timer is determined by the relative location of the feasible node to sink. The closer a node is to the sink, the shorter its backoff timer. This is to ensure that the node closest to the sink will send a CTS before a node that is farther away. Once a node has sensed another node's CTS, it discards its own. In this case, B has a shorter backoff period than A. As shown in Fig. 2, when B sends its CTS, A hears it. A realizes that another feasible node is closer to the sink, and thus, A goes to sleep. If A does not hear B's CTS and sends a CTS anyway, the data packet (DATA) sent by src for B clarifies which node is the chosen node. Note that the possibility exists that there is no node in the feasible region with an initiative $\mathcal{I} = 1$. In which case, XLM will retransmit. If the same situation arises, then XLM will switch to angle-based routing (ABR). A discussion of ABR is beyond the scope of this paper. ABR is described in [1].

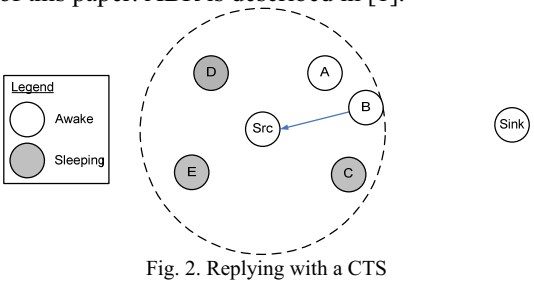

Fig. 2. Replying with a CTS

Once the source receives a CTS, it designates the CTS transmitter as the relay (or sink) and sends DATA. In this example, once src receives the CTS from B, it sends DATA to B, as shown in Fig. 3.

Once DATA is received, the relay (or sink) replies with an acknowledgement (ACK), as shown in Fig. 4. Otherwise, if DATA is not received by B, the data timer will expire, and the src will begin a retransmission.

Upon receipt of the ACK, all nodes go back to idling or sleeping. In this example, B begins a transmission of its own to transmit DATA towards the sink.

In [1], XLM is simulated on the cross-layer simulator (XLS) developed using C++. The authors document that XLM outperforms traditional layered protocol architectures, such as Flooding, GEO and PRR, in terms of network performance, energy consumption and complexity of implementation [1].

III. Protocol Details

A. Definitions and Assumptions

The following terms will be used in describing the integrated protocol CoopXLM. They are illustrated in Fig. 5-6 below.

Source (S) – The node that is generating or relaying the packet. The description of CoopXLM does not refer to the specifics of how the source gets its packet.

Cooperative nodes (CN) – The nodes that might be selected to send DATA in a cooperative transmission. These nodes are selected by the source and are a superset of cooperative buddies.

Cooperative buddies (CB) – The nodes that are selected to send DATA cooperatively. They work together to synchronously send DATA to a single destination. Cooperative buddies are a subset of cooperative nodes.

Cooperative leader – The cooperative buddy that is closest to the source. This node will begin the convergent transmission.

Destination (D) – The cooperative buddies will synchronously send DATA to this node. This node can be any node, including the sink. As long as this node is any node but the sink, after successfully receiving DATA, it will commence a divergent transmission of its own.

Sink – The final target node of all packets. This node only receives packets. It never transmits them.

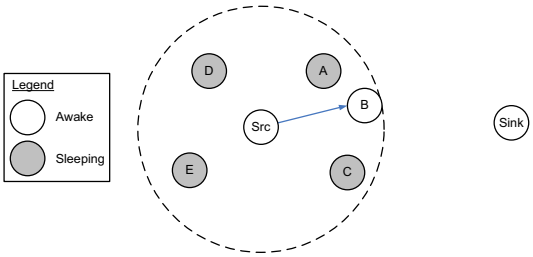

Fig. 3. Sending DATA to the chosen node

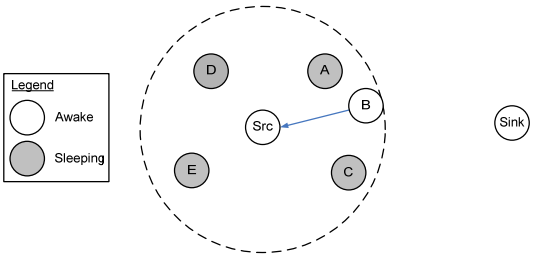

Fig. 4. Sending an ACK

Divergent transmission – The transmission of DATA from the source to cooperative buddies. It begins when a source sends a request to send (RTS) and ends when the cooperative buddies' acknowledgements (ACKs) are received at the source or when the source's clear to send (CTS) or ACK timeouts expire and the source retransmits. It is equivalent to broadcast mode in [3].

Convergent (cooperative) transmission – The transmission of DATA from cooperative buddies to a destination. It starts when all cooperative buddies transfer their shared packet to the priority queue and the cooperative leader sends out an RTS. It ends when the destination's ACK is received by all cooperative buddies or when either the CTS or ACK timeout expires and the cooperative buddies disband. It is equivalent to cooperative mode [3].

Cooperative hop – The transmission of data from the source to cooperative nodes to a destination. It consists of a divergent transmission followed by a convergent (cooperative) transmission.

Priority packet – A packet that is received in a divergent transmission and will be transmitted by cooperative buddies through a convergent transmission. It has priority over all of the packets in a cooperative buddy's First In First Out (FIFO) buffer.

Maximum number of cooperative nodes (m) – A parameter that limits the number of cooperative nodes for each divergent transmission. A value of 1 leads to XLM transmissions, i.e., no cooperation.

A few assumptions must be made in the integration of cooperation and XLM. First, each node should not only know its location and that of the sink, as required in XLM, but it should also know the positions of its neighbor nodes. Since this is a static network, it is easy for the nodes to establish their positions at the beginning of the deployment, and then share them with their neighbors once. This information is used by the source to determine the cooperative leader at the start of the convergent transmission.

In addition, nodes need to know their neighbors' channel conditions. This can be established either by a learning phase at the time of deployment or through the control packets, RTS, CTS and ACK.

Last, cooperative buddies must match their phases for coherent reception at the destination. An alternative to this requirement is that the destination has some circuitry for coherent reception.

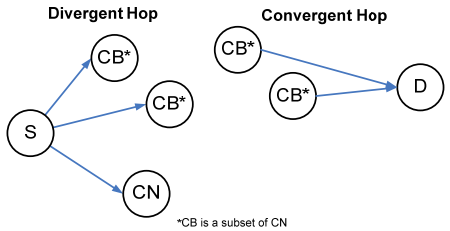

Fig. 5. Illustration of node definitions

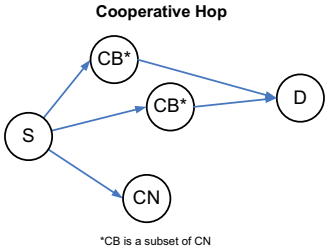

Fig. 6. Illustration of hop definitions

B. Unique Aspects of CoopXLM

CoopXLM has several key differences from XLM. In the most simplified state, CoopXLM modifies the receiver-based contention of XLM to allow more than one node to receive the data from a source. However, XLM was not created to handle multiple receiver nodes. For this reason, divergent and convergent modes are introduced. Both modes are defined above. Other unique aspects are a priority queue for each node; modified divergent CTS transmission and timeout; and modified divergent ACK transmission. Each of these modifications is described below. Then, the design is described in its entirety in the next subsection.

The priority queue is a new addition to the current sensor node definition. Presently, sensor nodes have memory in the form of a buffer to store the packets they generate and relay. The buffer in XLM processes packets using the first in first out (FIFO) method, which means that when a packet is generated or received, it will not be processed until after all packets already in the queue are processed. In CoopXLM, a priority queue is defined. The priority queue is a location in memory that can only store one packet. If there is a packet in the priority queue, it is called the priority packet and it is processed before any packets that are already in the buffer. The priority queue is only used by nodes that have established a set of cooperative buddies. The cooperative buddies place their common packet into each of their priority queues. In this way, all the cooperative buddies will be processing the same packet at once. If the cooperative buddies timeout and disband, this packet will be copied into the ordinary buffer and the priority queue will be cleared. Since a node can only be involved in one cooperative communication at a time, the priority queue's size of one is adequate.

The new ACK transmission is described in the following subsection. In the divergent hop, the ACK timeout is a function of the maximum number of cooperative nodes (m). Also in the divergent hop, more than one node will send a CTS as reply to the cooperative leader's RTS. In order to allow for more time for CTSs to be received, the CTS timeout has been extended as a function of the maximum number of cooperative nodes (m). In addition, the source will not transmit until its CTS timeout has expired. Since the CTS timeout is much longer than the CTS backoff timer, this condition ensures that DATA does not collide with other CTSs.

Each of the unique aspects of CoopXLM was described above. The following subsection describes the full CoopXLM protocol.

C. Protocol

The CoopXLM implementation works as follows. The source sends an RTS to all neighbor nodes. These nodes use receiver-based contention to decide which nodes are able to reply with a CTS. In the XLM protocol, if a node backing off to send a CTS receives a CTS from another node, it drops its CTS. In order to perform cooperation, the source node needs to select at least two relay nodes. For this reason, multiple nodes must reply with CTSs. CoopXLM does not allow nodes to stop backing off to send a CTS if they receive another node's CTS. In the XLM protocol, nodes are prioritized according to their proximity to the sink. Those that are closer to the sink will reply with a CTS before those that are farther away from the sink. This latter feature is incorporated into CoopXLM.

The source accepts the first CTS it receives. After that it ensures that subsequent CTS-senders are within the transmission radius of the initial CTS-sender. This condition is necessary to ensure that the cooperative nodes are able to communicate in the convergent transmission. The source keeps accepting CTSs as long as it does not have m cooperative nodes. Once the source has received and accepted m number of CTSs, it stops accepting CTSs and waits for its CTS timer to expire. Since the CTS timer is much longer than the CTS backoff timer, the source waits for its CTS timer to expire before sending DATA. This avoids the possibility of collision between DATA and CTSs. The CTS timer is also parameterized according to m. The nodes whose CTSs are not accepted know that they are not part of the communication when their DATA timers expire. All nodes whose CTSs were accepted are cooperative nodes.

Fig. 7 illustrates the CTS transmissions in CoopXLM. Assume that the sink is off to the right hand side of the destination, dest. Recall that all nodes know their positions and the positions of their neighbors. In this example, m = 4. Src has already sent an RTS for the divergent hop. Since B is closest to the sink and still within src's transmission radius, its backoff timer expires first. Thus, B sends its CTS to src first. Src accepts B as a cooperative node. Next, C sends a CTS followed closely by A. Src evaluates the distance between B and C to determine if C is within the transmission radius of B. Since C meets this criterion, it is accepted as another cooperative node. Src performs the same evaluation for A, but this time A must be within the transmission ranges of both B and C. Next, D's backoff timer expires, so it sends a CTS. D is not within the transmission radius of C, so it does not become a cooperative node. D goes to sleep when its DATA timer expires. At the end of this step, A, B and C have been chosen

as cooperative nodes for src.

The source stops accepting CTSs once the requisite number of nodes have been accepted as cooperative nodes. If no node has sent a CTS by the time the source's CTS timer expires, the source attempts to retransmit. Notice that even though m = four, in this example only three cooperative nodes are chosen. If at least one cooperative node has been selected, the source broadcasts the DATA to all cooperative nodes. Due to WBA, all cooperative nodes hear the DATA.

The header of the data packet contains a list of all of the cooperative nodes in the order in which they must send ACKs. The order of the cooperative nodes is determined by their distances to the sink. The node closest to the sink is slated to be first; the farthest cooperative node is last. The ACK transmission is a time division transmission. As the number of cooperative nodes increases, the ACK length increases linearly. Fig. 8 illustrates the ACK transmission resulting from the source (src) sending DATA with a header of (B, C, A). In this case, all cooperative nodes receive the DATA, so they each transmit an ACK in the appropriate time slot. Each node is able to listen to the ACKSs sent by the other cooperative nodes when it is not transmitting. If a node does not successfully receive DATA, it does not send an ACK; thus, its time slot is empty. No other slots change. This is illustrated in Fig. 9. Fig. 9 illustrates the ACK transmission resulting from the same source and cooperative nodes as in Fig. 8. However, in this case, C does not receive the data, so it does not participate in the ACK transmission. C awaits a DATA packet from src until its DATA timer expires. By the end of the divergent transmission, each cooperative node knows the other nodes it is cooperating with, which are now its cooperative buddies. Each cooperative buddy characterizes the DATA packet it just received as a priority packet. DATA is saved in each cooperative buddy's priority queue. Recall that only one packet can be in the priority queue at a time.

The initial XLM example will now continue as a CoopXLM one. For CoopXLM, Fig. 10 replaces Fig. 4. Since it is assumed that only cooperative nodes A and B receive data, A and B become cooperative buddies while C waits for its DATA timer to expire.

Once the source receives the ACKs, it designates the cooperative buddy farthest from the sink as the cooperative leader. The cooperative leader is chosen in this way in order to maximize the probability of the destination's CTS reaching all cooperative buddies. In the example in Fig. 10, A becomes the cooperative leader. Note that the source ends its participation in the communication with the end of the divergent hop.

The cooperative leader initiates the converging cooperative transmission by sending an RTS to its neighbor nodes. The RTS header contains a list of all of its cooperative buddies. The cooperative leader then awaits CTSs, which are sent the same way as in the original XLM. The other cooperative buddies do not send RTSs. They wait for the CTS replies to the cooperative leader's RTS. Only a CTS that is received by all the cooperative buddies is accepted. If no valid CTS is received within the CTS timeout period, all of the cooperative buddies disband. Disbanding involves each node

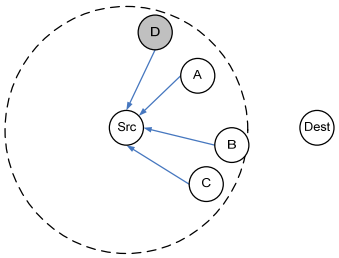

Fig. 7. CTS Transmission in the divergent hop

moving the priority packet from its priority queue to the buffer. Then each cooperative buddy proceeds to process the packets in its buffer as per the XLM protocol. The RTS that is sent out by each cooperative buddy initiates an individual divergent communication. Notice that due to the possible disbanding of cooperative buddies, duplicates of packets might propagate through the network.

Once all the cooperative buddies hear the same CTS, the CTS-sender is designated as the destination. In Fig. 11, the destination is labeled dest. Notice that src does not participate in the convergent hop, so it goes to sleep. The cooperative leader, A, is now the center of the transmissions. Dest has sent a CTS that is received by both cooperative buddies, A and B.

Each cooperative buddy computes the transmission power necessary for it to send the DATA packet using the maximal combining attenuations and weights shown in (2) and (3).

Next, all the cooperative buddies simultaneously transmit the priority packet to the destination. In this integrated protocol, both the average path loss and the instantaneous path loss are modeled. The former is learned through control packets, while the latter is a probabilistic value that models fading and multipath effects. Thus, there is no guarantee that a packet transmitted cooperatively will be received at the destination. The received SNR is given by (4). The goal is that the diversity from the cooperative transmission results in a higher received SNR, so more packets can be successfully received.

In the last step, the destination sends an ACK to all of the cooperative buddies. If the ACK is received successfully by each cooperative buddy, then the converging transmission and the cooperative hop is a success. If a cooperative buddy does not successfully receive the ACK before its ACK timer expires, it performs the disbanding procedure described earlier in this section.

IV. Performance Evaluation

In order to evaluate the CoopXLM protocol, a cross-layer simulation platform developed in [1] is used. In the simulations, the WSN consists of 300 Crossbow MICA2 nodes distributed randomly in a 100 m x 100 m field. The event occurs at (20 m, 20 m) and 10 nodes within the event radius of 20 m send data to the sink, which is located at (80 m, 80 m). Table I displays the simulation parameters. The optimal transmission range of the nodes is 39 m; although, the channel conditions limit how far the packet can travel and still be successfully received. The retransmission limit is typical of wireless networks. The receive, transmit, and sleep powers are modeled according to the MICA2 mote [5].

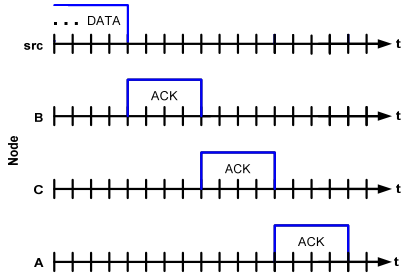

Fig. 8. ACK Transmission where all cooperative nodes become cooperative buddies

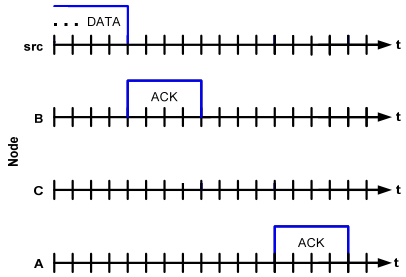

Fig. 9. ACK Transmission resulting where only some cooperative nodes become cooperative buddies

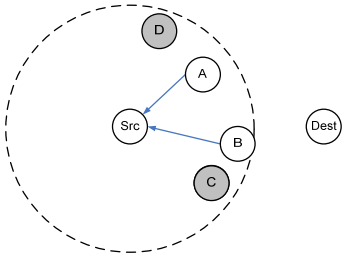

Fig. 10. ACK transmission in the divergent hop

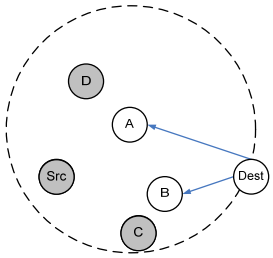

Fig. 11. CTS transmission in the convergent hop

The maximum transmit power is defined as 25.5 mW in Table I. A node uses that amount of energy during non-cooperative communications, but when cooperating, the node's power can vary. The experimental results from [7] have been used to create a more accurate model of the transmit energy required. Using the desired transmit power, the current used in the transmit circuitry is interpolated. The power is defined as the current drawn from the power supply multiplied by the 3 V power supply. For example, if 1 mW is transmitted, then the current drawn is 8.5 mA, which yields a total transmission power of 25.5 mW. The energy consumed in 1 second will be 25.5 mJ. If the transmission uses half the power, then the current drawn is 6.67 mA, which yields a total transmission power of 20 mW. The energy consumed in 1 second will be 20.0 mJ. It is clear that the total transmission energy is non-linear and that by halving the transmission power, the total transmission energy is not simply halved.

$$\alpha_i = \frac{1}{\sqrt{PL_{avg}(coopBuddy(i), dest)}} \tag{2}$$

$$w_i = \frac{\alpha_i}{\sqrt{\sum_{i=1}^{n} \alpha_i^2}} \tag{3}$$

$coopBuddy$ = An array of the cooperative buddies
$dest$ = Destination node
α_i = Attenuation between cooperative buddy i and the destination
w_i = Weight to yield maximal SNR
PL_{avg} = Average path loss component between a cooperative buddy and the destination
n = Number of cooperative buddies

$$SNR_{rvcd}(j) = \frac{P_t}{P_n} \left\{ \frac{\sum_{i=1}^{n} \frac{1}{\sqrt{PL_{inst}(coopBuddy(i), dest)}}}{\sqrt{\sum_{i=1}^{n} \frac{1}{PL_{avg}(coopBuddy(i), dest)}}} \right\}^2 \frac{1}{PL_{avg}(j, dest)}$$

$$SNR_{total} = \sum_{j=1}^{n} SNR_{rcvd}(j) \tag{4}$$

$SNR_{rcvd}(j)$ = Received SNR at the destination from cooperative buddy j
SNR_{total} = Total received SNR at the destination from all the cooperative nodes
PL_{inst} = Instantaneous path loss component between a cooperative buddy and the destination
PL_{avg} = Average path loss component between a cooperative buddy and the destination
P_t = Transmitted signal power
P_n = Noise power
n = Number of cooperative buddies
$coopBuddy$ = An array of the cooperative buddies
$dest$ = Destination node

A. Simulation Performance Measures

It can be difficult to establish relevant metrics to determine if the network is benefiting from cooperation [4]. The metrics that are measured are average goodput (%), average consumed energy per packet (J), average unique goodput (%), average number of hops per packet, average latency per packet (s) and the average number of successful convergent hops. The goal of the metrics is to compare CoopXLM to XLM. Each metric is described below followed by a prediction of how CoopXLM will compare to XLM.

There are two types of throughput graphs: goodput and unique goodput. The goodput is the percentage of packets injected into the network by the source nodes that are received at the sink. In a cooperative network, duplicates of packets will be propagated if cooperative buddies disband. If more than one copy of a packet is received at the sink, the goodput can actually be greater than 1. While the goodput includes the duplicate packets received at the sink, the unique goodput discards duplicate packets received and displays only the percentage of unique packets that were received. The unique goodput can never be greater than 1. For a non-cooperative (XLM) transmission, there are no duplicate packets, so the

TABLE I
SIMULATION PARAMETERS

Parameter	Value
Retransmit limit	7
Transmission rate throttle factor (β)	2
Linear packet rate adjuster (α)	Initial packet rate / 10
Buffer length	30
Length of control packet	20 bytes
Length of data packet	100 bytes
Frame length	5 s
Energy Threshold (E_{rem}^{min})	100 μJ
SNR Threshold (ξ_{Th})	5 dB
Transmitted power (P_t)	5 dB
Power level for the reference distance (PL(d_0))	55 dB
Noise power (P_n)	-105 dB
Path loss exponent (n)	3
Variance of shadowing (σ)	3.8
Coherence time ($T_{coherence}$)	100 ms
Receive energy (E_{rx})	13.5 mW
Transmit energy (E_{tx})	25.5 mW
Sleep energy (E_{sleep})	15 μW
Simulation time	100,000 tics
Maximum number of cooperative nodes (m)	4

goodput and unique goodput are identical. It is predicted that the goodput of CoopXLM should remain constant when compared to XLM. This is due to the increased diversity of the cooperative transmission and the increased opportunity of packets to be received at the sink in the case that the cooperative buddies disband. However, there is also the possibility of increased collisions due to the increase in packet traffic and longer transmission times.

Average energy is calculated using the total number of packets received at the sink because all packets and duplicate packets use energy. Dropped packets are not counted as packets in the calculation because although their propagation consumed energy, they are the price to pay to receive the successful packets. Since there can be duplicates of packets, this implies that there are really two ways to calculate average energy. Say that each hop takes 1 J of energy. In this case, both the original packet's energy and the duplicate packets' energy are 4 J because they each take 4 hops to get to the sink. The two different methods of calculating average energy per packet are shown in (5) and Fig. 12. The second method overestimates the average energy per packet, by attributing all the energy for the duplicate packets to the original packet. The first method is a fair way of dividing the resource consumption. Notice that the 1 J from the initial hop is not counted twice in the energy calculation. With the use of cooperative communication in CoopXLM, it follows that the average consumed energy per packet should decrease.

The average latency per packet is the average time it takes for a packet to be injected into the network to when it is received at the sink. The injection time is recorded when the source begins backing off to send an RTS. Since duplicate packets will have valid start and end times, their latencies are also counted. In a cooperative network, the CTS timeout and ACK duration are increased significantly. The source node must wait for its CTS timeout to expire before it sends any DATA in order to reduce the chance of colliding with CTSs. The CTS timeout is a function of m. As m increases, so should the latency. The time it takes to send ACKs in the divergent

$$E_{avg1} = \frac{E_{src,A} + E_{A,B,C,Sink} + E_{A,D,E,Sink}}{2pkts}$$
$$E_{avg2} = \frac{E_{src,A} + E_{A,B,C,Sink} + E_{A,D,E,Sink}}{1pkt} \quad (5)$$

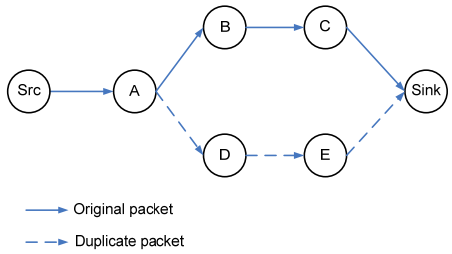

Fig. 12. Duplication of a packet

transmission is a function of the number of cooperative nodes involved in that particular communication. Due to the increases in these timers, it follows that for the average latency of a packet should increase.

A single cooperative communication consists of two hops: the divergent transmission and the convergent transmission. The average number of hops counts a full cooperative communication as 2 hops. Note that hops are only counted once even when duplicate packets are included. In Fig. 12, even though both the packet and its duplicate will have counted 4 hops, the metric only registers 4 hops for the original packet and 3 hops for the duplicate. Thus, the total number of hops for this packet and its duplicate is 7. It follows that the average number of hops for a packet should be comparable in both the non-cooperative and the cooperative cases.

The percentage of convergent hops is calculated by dividing the number of successful convergent hops by the total number of successful convergent and divergent hops. The percentage of convergent hops will be 0% for XLM because it does not use cooperative communication.

B. Results

In this section, the results of two sets of simulations are shown. In the first set, m is varied from 1 to 10, where 1 is the XLM case. The objective is to see if there are energy savings when comparing CoopXLM to XLM. The second set is run while varying SNR threshold values from -10 dB to 10 dB for both XLM and CoopXLM. The objective is to investigate whether using cooperation decreases the network's sensitivity to SNR threshold.

Goodput was calculated but is not shown because goodput includes duplicates. The unique goodput is the more important of the two measures because it determines how many unique packets reached the sink. In Fig. 13, the unique goodput is in the range of 65% and 92% for a 10% duty cycle The results for all cooperative networks generally rise then fall as the duty cycle increases from 10% to 100%. The initial increase is due to the availability of more neighbor nodes as relays. Then, as the duty cycle increases above 50%, there is a noticeable decrease in the unique goodput. This change is due to the increased collisions caused by more nodes being awake

for longer. In CoopXLM, the CTS and ACK transmission times are longer, which means there are more opportunities for collisions. It is possible that nodes are retransmitting several times during a single CTS or ACK transmission. The possibility of this occurring increases as m increases.

It is interesting to note that the non-cooperative results are in between the cooperative results for duty cycles from 10% to 50%. It is only for duty cycles above 50% that the non-cooperative case has a better unique goodput than the cooperative cases. The case where m = 2 seems to be competitive with the non-cooperative case in terms of unique goodput.

The energy consumed per packet is the central focus of this paper. Fig. 14 displays the energy consumption results. It is clear that the non-cooperative case uses more energy than any of the cooperative cases. While the lowest average consumed energy per packet for the non-cooperative case is 0.21 for a 10% duty cycle, the most energy consuming cooperative case uses only 0.15. The cooperative cases use from 12% to 55% less energy per packet than the non-cooperative case. The average energy savings over all duty cycles is 38%.

On average, the trend is that as m increases, the energy consumption decreases. This is clear if the case where m = 2 is compared to m = 9. However, the case of m = 10 is contrary to the trend. This contrary result might be explained in future research by tracking the actual number of cooperative buddies being used. The general upward trend in energy consumption over the duty cycles can be attributed to more nodes being awake; and thus, more nodes sending CTSs.

Fig. 15 shows the percentage of convergent transmissions. As expected, the non-cooperative, or XLM, case has no converging transmissions. In the cases where m = 2 to m = 5 stay between 9% and 20% for all duty cycles. The results shrink to 6% to 13% for the cases where m = 6 to m = 10. It is unexpected but interesting to note that as m decreases, the percentage of convergent transmissions increases. It is unclear why this occurs because convergent transmissions can occur even if the number of cooperative buddies is less than m. One possible explanation is that it is difficult to find more than one node that can send a CTS.

Fig. 16 shows that as the duty cycle increases across all cases, the latency is reduced. This can be attributed to more nodes being available to relay packets. In the cooperative cases, it can be attributed to more nodes being available to cooperate. As expected, latency increases as m increases. This was expected because the CTS and ACK transmissions are linearly increasing as m increases. It is interesting to note that the latency is not significantly higher for the case where m = 1 compared to the case where m = 2. The case where m = 9 has the highest average latency and the lowest average energy consumed. There seems to be a trade-off in these measures.

The latencies of just the packets that went through at least one convergent hop were plotted, as were the latencies of those packets that did not go through any convergent hops. These plots were omitted for brevity since they were very similar to Fig. 16.

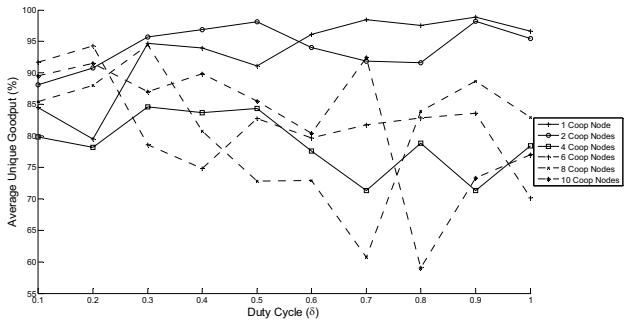

Fig. 13. Unique goodput

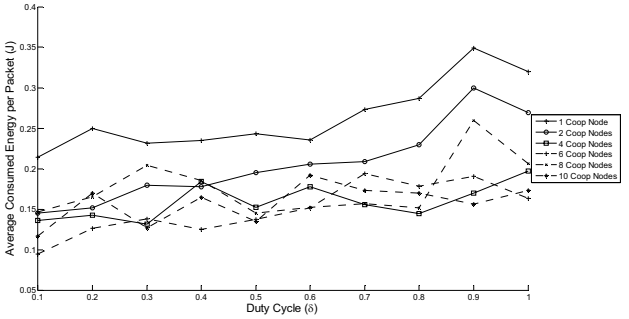

Fig. 14. Consumed energy

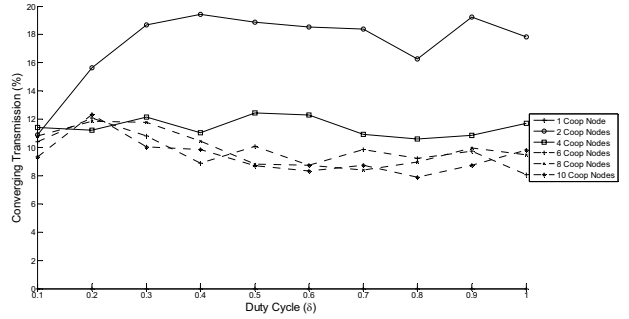

Fig. 15. Percentage of convergent transmissions

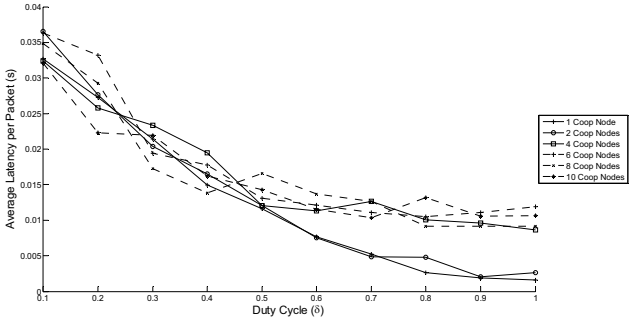

Fig. 16. Average Latency

Fig. 17 shows the average hops per packet. The non-cooperative case has fewer average hops than CoopXLM. This might be attributable to the position of the cooperative leader. The cooperative leader was chosen to be the cooperative buddy farthest from the sink in order to increase the probability that all cooperative buddies would receive the destination's CTS. This works reasonably well if m is small because there are enough nodes that are near the transmission

range limit of the source. However, for larger values, the probability of the source having that number of nodes near its transmission range limit is small. For this reason, nodes that are very close to the source are selected and the hop sizes become smaller. Future research might find it interesting to track the average hop size as related to the actual number of cooperative nodes in a transmission.

Another analysis was done by varying the SNR thresholds from 10 dB to -10 dB for XLM and CoopXLM. For the cooperative case, m = 4.

Fig. 18-21 show that XLM has a more predictable reaction to changes in SNR thresholds than CoopXLM. There appears to be no advantage to using CoopXLM over varying SNR thresholds. XLM is more sensitive to duty cycles than it is to SNR thresholds. At duty cycles below 40%, XLM has lower unique goodput. In fact, SNR thresholds affect XLM more when duty cycles are low. This is because lowered duty cycles reduce the number of nodes that are awake, while higher SNR thresholds reduce the number of nodes that meet the initiative determination and thus are able to backoff to send a CTS. With those effects combined, the probability of finding a relay node decreases drastically, so unique goodput decreases. Unlike XLM, CoopXLM displays higher unique goodput as the duty cycle decreases. This might be because at low duty cycles, CoopXLM does not have congestion. It would be interesting to determine which of the initiative determination criteria are most influential in weeding out feasible nodes in both XLM and CoopXLM.

The data from Fig. 18-21 is plotted as a function of SNR thresholds in Fig. 22-24. As duty cycle increases, XLM has a higher goodput across all SNR thresholds. This is in concordance with the previous discussion.

V. CONCLUSION

Wireless communication systems have a multitude of potential applications; however, one serious disadvantage of wireless sensor networks is that the sensors used in these networks are often limited to the use of a single battery, and their success is highly dependent upon power efficient protocols. Traditional layered protocols often prove to be inefficient for WSNs. One recently developed protocol, the cross-layer module (XLM), is a unified protocol that is designed specifically for WSNs. XLM has been shown to increase network efficiency and reliability in comparison to traditional layered protocols. The core of XLM, initiative determination, maintains a balance between received signal-to-noise ratio (SNR), local congestion and remaining energy to increase reliability and network lifetime. Cross-layer design can help achieve better energy performance in WSNs; however, it is important to strive for the advantages of flexibility, modularity, simplicity and scalability found in traditional layered protocols [2].

Cooperative communications is also an alternative to traditional protocols that can increase efficiency in WSNs by having several nodes simultaneously transmit a single message to the intended destination. In this paper, a new protocol termed CoopXLM, which integrates cooperative communication with XLM, is created and examined via simulation. Simulation results indicate that energy is saved with CoopXLM in comparison to XLM and the CoopXLM yields higher goodput at lower duty cycles but yields lower goodput at higher duty cycles. CoopXLM is a protocol that should be used when power savings is of paramount importance, such as in the case of WSNs.

In future studies, it would also be interesting to use cooperative communication for increased hop length. Instead of varying transmission power, the cooperative nodes would instead transmit simultaneously at full power. The transmitted signal would be able to go farther for the same received SNR. This scheme would decrease the number of hops required for a packet to reach the sink; thus, the power used to propagate a packet through the entire network would decrease. This scheme also has its challenges. For example, using the XLM protocol, how would the cooperative nodes be able to select a common destination node? There is no guarantee that the nodes that heard the cooperative RTS would be able to send a CTS with an SNR that would be received by all the cooperative nodes.

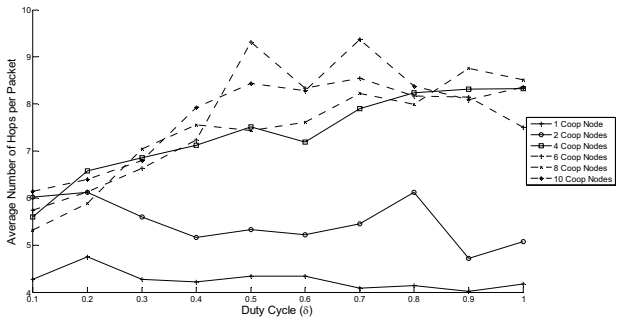

Fig. 17. Average hops

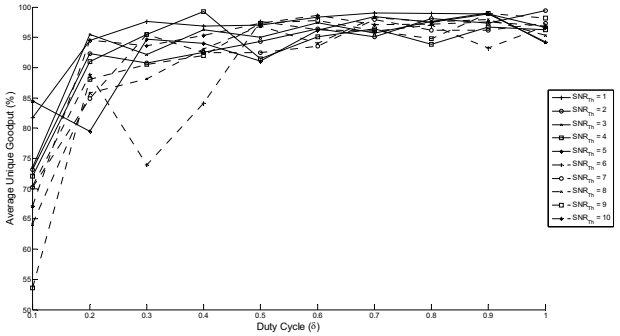

Fig. 18. Unique goodput for SNRs from 1 dB to 10 dB for m = 1 (XLM case)

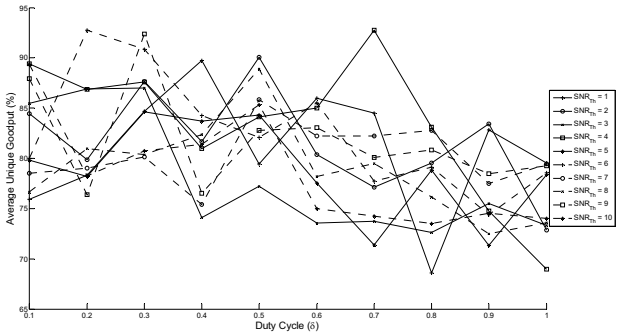

Fig. 19. Unique goodput for SNRs from 1 dB to 10 dB for m = 4 (CoopXLM)

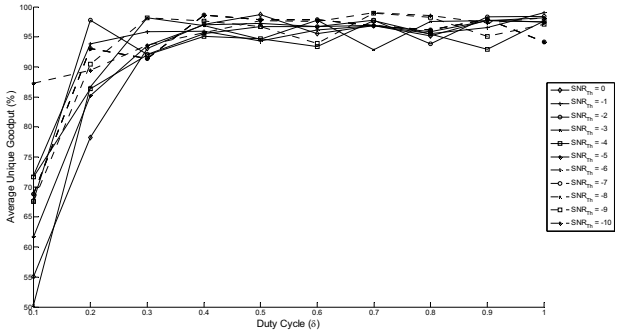

Fig. 20. Unique goodput for SNRs from -10 dB to 0 dB for m = 1 (XLM case)

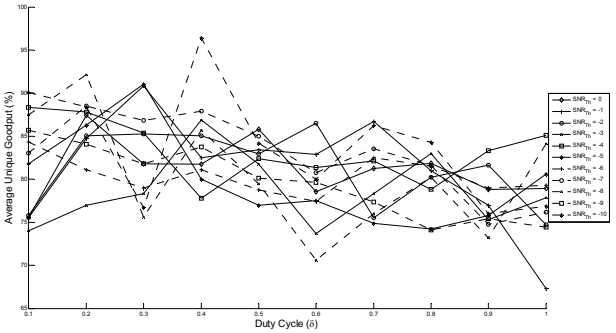

Fig. 21. Unique goodput for SNRs from -10 dB to 0 dB for m = 4 (CoopXLM)

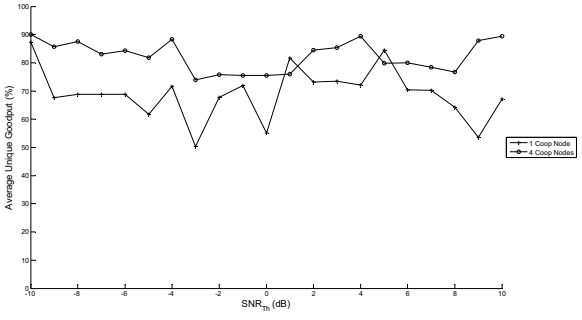

Fig. 22. Unique goodput for duty cycle is 10% across all SNRs for m = 1 (XLM case) and m = 4 (CoopXLM)

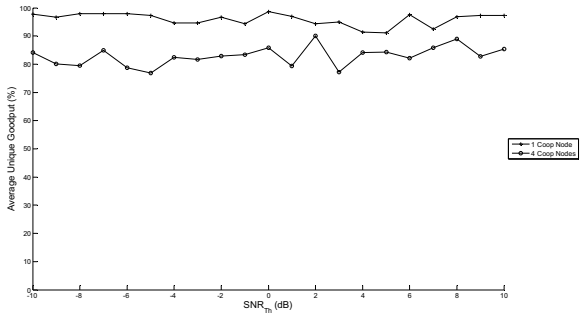

Fig. 23. Unique goodput for duty cycle is 50% across all SNRs for m = 1 (XLM case) and m = 4 (CoopXLM)

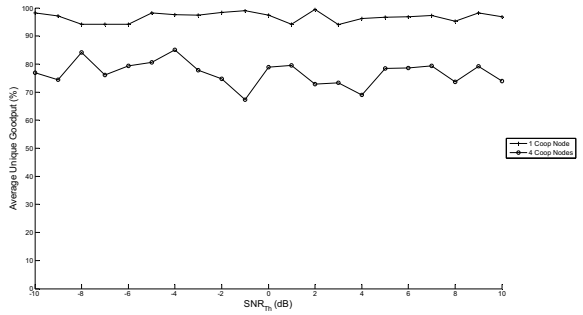

Fig. 24. Unique goodput for duty cycle is 100% across all SNRs for m = 1 (XLM case) and m = 4 (CoopXLM)

REFERENCES

[1] Akyildiz, Ian F., Mehmet C. Vuran, and Ozgur B. Akan, "A Cross-Layer Protocol for Wireless Sensor Networks," *40th Annual Conference on Information Sciences and Systems*, pp. 1102-1107, 22-24 March 2006.

[2] Cui, Shuguang and Andrea J. Goldsmith, "Cooperation Techniques in Cross-Layer Design," Fitzek and Katz, pp. 101-126.

[3] Khandani, Amir Ehsan, Jinane Abounadi, Eytan Modiano, and Lizhong Zheng, "Cooperative Routing in Static Wireless Networks," *IEEE Transactions on Communications*, Vol. 55, No. 11, pp. 2185-2192, November 2007.

[4] Mähönen, Petri, Marina Petrova and Janne Riihijärvi, "Cooperation in Ad-Hoc Networks," Fitzek and Katz, pp. 189-222.

[5] "MICA2," Crossbow Technology, Inc. 19 November 2008 <http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf>.

[6] "MICAz," Crossbow Technology, Inc. 19 November 2008 <http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAZ_Datasheet.pdf>.

[7] Shnayder, Victor, Mark Hempstead, Borrong Chen, Geoff Werner Allen, and Matt Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications," Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys '04), pp. 1-13, 3-5 November 2004.

[8] Vuran, Mehmet C. and Ian F. Akyildiz, "Cross-Layer Analysis of Error Control in Wireless Sensor Networks," 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON), pp. 585-594, 28 September 2006.

[9] Zuniga, Marco and Bhaskar Krishnamachari, "Analyzing the Transitional Region in Low Power Wireless Links," 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004, pp. 517-526, 4-7 October 2004.